Efficient 3D video streaming

Petrovic, G.

DOI:
10.6100/IR754838

Published: 01/01/2013

Citation for published version (APA):
Efficient 3D Video Streaming

Proefschrift

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 1 juli 2013 om 16.00 uur

door

Goran Petrovic

geboren te Nis, Servië
Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. P.H.N. de With

A catalogue record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-3396-1
Summary

3D-TV based on stereo vision has been gradually introduced recently and was enabled by the availability of the first generation of 3D-TV displays and the storage of stereo-formatted movies on BluRay disc. The emerging application domains for 3D video include, besides TV entertainment, communications, medicine, security, visualization and education. In each of these domains, 3D video brings specific advantages compared to conventional 2D video. In general, these advantages yield a sense of immersion, realism of the presentation and particularly for entertainment, enhanced viewing experience. The two main application scenarios in entertainment and advanced video communications are Three-Dimensional Television (3D-TV) and video and Free-Viewpoint Television (FTV) and video communications. In this thesis, we present efficient algorithms and architectures for these application scenarios in the form of services, while focusing on efficient 3D video transmission and delivery aspects. Our research scope includes efficient 3D video data representations, efficient 3D video compression techniques and efficient video streaming algorithms and delivery architectures, where we concentrate on system aspects like limited and varying Internet bandwidth, latency and interactivity. The thesis contributes as follows.

In Chapter 2, we present three virtual-view rendering algorithms that employ different 3D video representations to significantly reduce the bandwidth requirement of a 3D video streaming system at a negligible penalty in rendering quality. The first and second algorithm employ 3D video representations that combine textures with scene-geometry models in the form of depth and disparity maps, respectively. They achieve a high rendering quality by relying on principles of multiview geometry of a 3D scene to guide the rendering process. These algorithms are not our original contributions, but they form the rendering basis for streaming algorithms developed in subsequent chapters of this thesis. Next, we
propose an algorithm that renders undersampled light-field representations of 3D scenes at a high visual quality. This is achieved by minimizing rendering artifacts caused by aliasing in the synthesized views. The proposed algorithm - referred to as region-based all-in-focus light-field rendering - incorporates image segmentation to dynamically adapt the light-field rendering to scene depth-complexity. As a result, it renders all-in-focus virtual views of a 3D scene at a quality visually indistinguishable from the original views.

In Chapter 3, we address an important requirement for a 3D video streaming system, which is to simultaneously accommodate receivers that have heterogeneous resources or preferences. This chapter proposes a layered framework for 3D video streaming as a unifying and efficient solution to the problem of heterogeneity of receiving devices and viewing preferences. The framework is general in its applicability to unicast and multicast network architectures. The architecture components of our framework include: (1) efficient 3D video representation, (2) efficient decomposition of the 3D-scene description into information layers, where each layer conveys a single coded video signal or coded scene-geometry data, and (3) rendering of virtual views accommodating a layered scene description. Heterogeneous receivers can select the number of layers to receive for view rendering, depending on the availability of resources, or viewing preferences. To demonstrate the viability of the proposed architecture, we implement a 3D video streaming prototype and show that heterogeneous autostereoscopic 3D displays can be supported by the system. The prototype shows that the proposed layered streaming enables the system to scale the rendering quality with resource availability. Besides this, the proposed selective view streaming with interactive feedback accommodates the heterogeneity of viewing preferences.

In Chapter 4, we propose an algorithm for bandwidth-efficient 3D video streaming over a best-effort Internet. The proposed algorithm offers a continuous streaming and achieves a high rendering quality, despite the variations of available bandwidth common to best-effort networks. The main contribution in this chapter is to demonstrate that quality-optimized 3D video streaming algorithms should: (1) adapt the output streaming rate by explicitly considering rendering quality, (2) employ an adaptation of the underlying 3D video data representation, the coding algorithm and the rendering algorithm. To this end, we implement the proposed algorithm on the basis of existing 3D video representations and state-of-the-art algorithms for virtual-view rendering featuring the following key properties: (1) streaming rate allocation based on an optimized joint texture-depth rate allo-
tion and (2) virtual-view streaming adaptation that minimizes quality variations of computed allocations over time. The algorithm performance is evaluated using realistic simulations of Internet transmission conditions, including the impact of competing Internet traffic and real-world protocol implementations (e.g., ns-2). The results demonstrate a significantly higher average video quality over quality-agnostic rate control of texture and depth and conventional streaming strategies that are conservative in bandwidth usage.

In Chapter 5, we propose a 3D video streaming algorithm that achieves a low interaction latency and high rendering quality, even in streaming systems with large and time-varying system delay. Our main contribution is to demonstrate that future 3D video streaming systems should employ streaming algorithms that: (1) explicitly minimize user-perceived latency, (2) adapt to navigation patterns of the user and the available bandwidth and (3) explicitly control rendering quality during 3D-scene navigation. The proposed algorithm achieves these properties as follows. First, it prefetches texture and depth streams in order to reduce the latency perceived, while a user is switching between multiple views of a 3D scene. To optimize the prefetching rate, we analytically derive a user-navigation model and use this model to estimate the required streams and minimize unnecessary prefetching. Second, the algorithm adapts the 3D video streaming rate - including the prefetching rate - to increase its bandwidth efficiency on Internet paths with time-varying delay and bandwidth. Third, the algorithm minimizes quality variations among multiple, consecutively rendered views of the 3D scene by minimizing the distortion differences between those views. We implement the proposed algorithm in a network simulator and demonstrate that it achieves a given low target delay, provides smooth view transitions and maximizes the rendering quality. We also provide a visual evaluation of the rendering results to show that we achieve a sufficiently high perceptual rendering quality.

In Chapter 6, we propose an architecture for the delivery of multiview 3D video streams to a large number of concurrent users. The proposed architecture is a streaming Content Delivery Network (streaming-CDN) that provides the following services: rendering of virtual views, real-time encoding and streaming. The main insight of our proposal is that the conventional wisdom of regarding a streaming system as a distributed application with distributed data and centralized computation may not be an appropriate model for future multiview 3D video streaming systems. We argue that the alternative view of a multiview 3D video streaming system as an application with distributed data and distributed
computation is a better model for cost-effective realizations of a large system with resource-constrained and heterogeneous users. Specifically, our hypothesis in this chapter is that offloading the view-rendering computation to a remote location and providing it as a service of the existing streaming-CDNs is both technologically possible and useful for cost optimizations in 3D video streaming. To support this statement and to validate the hypothesis, our main contribution consists of: (1) analysis of the usefulness of the proposed architecture in the context of resource costs of today’s streaming-CDNs and (2) implementation of a small-scale 3D video streaming prototype according to the remote-rendering architecture. We have found that the proposed architecture reduces network bandwidth and that the implementation proved to be technologically feasible.
Samenvatting


In hoofdstuk 2 presenteren we drie rendering (weergave) algoritmen voor virtuele kijkhoek (view) in 3D video, die gebruik maken van verschillende datarepresentaties. Deze algoritmen verlagen de vereiste bandbreedte van een 3D-video streaming systeem aanzienlijk, ten koste van een verwaarloosbare verlaging van de weergavekwaliteit. Twee van de drie algoritmen gebruiken een datarepresentatie die de beeldtextuur combineert met geometrische modellen in de vorm van diepte en dispariteit (ongelijkheid tussen views) van de scène. Deze algoritmen bereiken een hoge weergavekwaliteit door het gebruik van geometrische principes uit mul-
**Samenvatting**

Tiview video bij de sturing van het beeldreconstructie proces. Deze algoritmen zijn niet onze eigen bijdragen, maar vormen de basis van de beeldreconstructie van streaming algoritmen die in de volgende hoofdstukken van dit proefschrift worden ontwikkeld. Vervolgens presenteren we een algoritme dat onderbemosteerde lichtrepresentaties van 3D-scènes met een hoge beeldkwaliteit reconstrueert. Dit wordt bereikt door het minimaliseren van renderingsartefacten die worden veroorzaakt door vouwcomponenten (aliasing) in de gesynthetiseerde beelden. Het algoritmevoorstel, aangeduid als regiogebaseerd all-in-focus lichtveld rendering, gebruikt beeldsegmentatie om de lichtveld rendering dynamisch aan te passen aan de dieptecomplexiteit van de scène. Het resultaat is een algoritme dat de all-in-focus virtuele views van een 3D-scène reconstrueert met een kwaliteit die visueel niet te onderscheiden is van de oorspronkelijke beelden.

In hoofdstuk 3 behandelen we een belangrijke voorwaarde voor een 3D-video streaming systeem, namelijk het gelijktijdig aanpassen van ontvangers die heterogene middelen of voorkeuren hebben. Het hoofdstuk beschrijft een gelaagd raamwerk voor 3D video streaming, dat fungeert als een uniforme en efficiënte oplossing voor het probleem van de heterogeniteit van ontvangers en de voorkeuren van kijkers voor bepaalde kijkhoeken. Het raamwerk is algemeen toepasbaar voor zogenaamde unicast en multicast netwerkarchitecturen. De architectuurcomponenten van het raamwerk zijn: (1) een efficiënte 3D-video representatie, (2) een efficiënte decompositie van de 3D-scène beschrijving in verschillende informatielagen, waarbij elke laag een gecodeerd videosignaal of gecodeerde geometrische gegevens van de scène transporteert, en (3) rendering van virtuele views door een gelaagde scènecbeschrijving. Heterogene ontvangers kunnen het aantal te ontvangen lagen voor view rendering selecteren, afhankelijk van de beschikbaarheid van hun middelen of voorkeuren in kijkhoeken. Voor validatie van de voorgestelde architectuur implementeren we een 3D-video streaming prototype en tonen aan dat heterogene autostereoscopische 3D beeldschermen kunnen worden gebruikt in het systeem. Het prototype laat zien dat de gelaagde streaming architectuur het mogelijk maakt om de reconstructiekwaliteit van het systeem op te schalen met de beschikbaarheid van de rekenkracht van de ontvanger. Daarnaast kan de voorgestelde architectuur voor streaming van de selectieve view en interactieve feedback zich aanpassen aan de heterogeniteit van gebruikte kijkvoorkeuren.

Hoofdstuk 4 beschrijft een algoritme voor 3D-video streaming met bandbreedte-efficiëntie voor een best-effort internet. Het voorgestelde algoritme biedt continue streaming en bereikt een hoge renderingskwaliteit, ondanks de variaties in
beschikbaarheid van de bandbreedte die typerend is voor best-effort netwerken. De belangrijkste bijdrage in dit hoofdstuk is het demonstreren dat de kwaliteitgeooptimaliseerde 3D-video streaming algoritmen moeten voldoen aan de volgende voorwaarden: (1) de uitgangssnelheid van streaming aanpassen zodat deze expliciet afhankelijk wordt van de renderingskwaliteit, (2) aanpassen van de algoritmen voor de onderliggende 3D-video datarepresentatie, het compressiealgorithmie en het renderingsalgorithmie. Daartoe implementeren we het voorgestelde algoritme op basis van bestaande 3D-video datarepresentaties en state-of-the-art algoritmen voor virtuele-view rendering, met de volgende belangrijke eigenschappen: (1) toewijzing van streaming bandbreedte gebaseerd op gezamenlijke bitrate-optimalisatie van textuur en diepte, (2) aanpassing van virtual-view streaming zodat de kwaliteitsvariaties van de berekende toewijzingen worden geminimaliseerd. De prestatie van het algoritme wordt beoordeeld met behulp van realistische simulaties van internettransmissie, met inbegrip van de invloed van concurrerend internetverkeer en werkelijk toegepaste protocolimplementaties (bijv. ns-2). De resultaten tonen een significant hogere gemiddelde videokwaliteit dan bij kwaliteitsdiagnostische controle van de data overdrachtssnelheid voor de textuur en diepte, en conventionele streaming strategieën die conservatief zijn in bandbreedtegebruik.

In hoofdstuk 5 stellen we een 3D-video streaming algoritme voor dat een lage interactievertraging en een hoge renderingskwaliteit bereikt, zelfs in streaming systemen met grote en tijdsvariabele systeemvertraging. Onze belangrijkste bijdrage is om te demonstreren dat toekomstige 3D-video streaming systemen streaming algoritmen moeten gebruiken die: (1) expliciet de door de gebruiker waargenomen vertraging minimaliseren, (2) zich aanpassen aan navigatiegewoonten van de gebruiker en de beschikbare bandbreedte, en (3) de renderingskwaliteit tijdens 3D-scène navigatie expliciet beheersen. Het voorgestelde algoritme bereikt deze eigenschappen als volgt. Ten eerste worden de textuur en dieptestromen van tevoren opgehaald (data prefetching), terwijl de gebruiker tussen verscheidene weergaven van een 3D-scène schakelt, om hiermee de waargenomen vertraging te verminderen. Voor de optimalisatie van de snelheid bij het vooraf ophalen van data, hebben we een gebruikersvriendelijk navigatiemodel analytisch afgeleid. Dit model wordt gebruikt om de vereiste streams te schatten, en voor het minimaliseren van onnodig ophalen van data. Ten tweede past het algoritme de 3D-video streaming snelheid aan - inclusief de prefetching snelheid - om hiermee de bandbreedte-efficiëntie te verhogen op internetpaden die een tijdvariërende vertraging en bandbreedte hebben. Ten derde minimaliseert het
algoritme kwaliteitsvariaties tussen meerdere, opeenvolgend weergegeven views van de 3D-scène door minimalisatie van vervormingsvariaties tussen deze views. We implementeren het nieuwe algoritme in een netwerksimulator en tonen aan dat het een gegeven lage doelvertraging bereikt, het zorgt voor soepele overgangen in kijkhoek, en het de renderingskwaliteit maximaliseert. We geven ook een visuele beoordeling van de renderingsresultaten om te laten zien dat we een voldoende hoge perceptuele renderingskwaliteit bereiken.

Hoofdstuk 6 beschrijft een architectuur voor de levering van multiview 3D-video streams naar een groot aantal gelijktijdige gebruikers. De voorgestelde architectuur is een streaming Content Delivery Network (streaming-CDN) dat de volgende diensten levert: reconstructie van virtuele views, real-time codering en streaming. Het belangrijkste inzicht van ons voorstel is dat de conventionele beschouwing over een streaming systeem als een gedistribueerde toepassing met gedistribueerde data en gecentraliseerde berekening, mogelijk geen geschikt model vormt voor toekomstige multiview 3D-video streaming systemen. We poneren dat het alternatieve model van een multiview 3D-video streaming systeem als een toepassing met gedistribueerde data en gedistribueerde berekeningen, een beter model is voor kosteneffectieve realisaties van een groot streaming systeem voor gebruikers met beperkte middelen en heterogeniteit in platform en gebruik. Meer specifiek is onze hypothese in dit hoofdstuk dat het uitvoeren van de view-rendering berekening bij een extern gepositioneerde (remote) gebruiker en het beschouwen hiervan als een dienst van de bestaande streaming-CDN’s, zowel technologisch mogelijk als nuttig is voor kostenoptimalisaties in 3D-video streaming. Om deze verklaring te ondersteunen en om de hypothese te valideren, bestaat onze belangrijkste bijdrage uit: (1) analyse van het nut van de voorgestelde architectuur in de context van kosten voor rekenkracht van de hedendaagse streaming-CDN’s, en (2) de implementatie van een kleinschalig prototype van een 3D-video streaming systeem volgens de remote-rendering architectuur. We hebben geconstateerd dat de voorgestelde architectuur de benodigde bandbreedte reduceert, en dat de uitvoering technisch haalbaar is.
# Contents

Summary i

Samenvatting v

1 Introduction 1

1.1 Preliminaries on 3D video 1
1.2 3D video communication system 3
1.3 Research scope and assumptions 4
1.4 Overview of 3D video system technologies 5
  1.4.1 3D video data-representations 6
  1.4.2 3D video compression 8
  1.4.3 Video streaming algorithms and delivery architectures 9
1.5 Problem statement and research questions 11
1.6 Research contributions 14
1.7 Thesis outline and publication history 16

2 Efficient 3D Video Representations and Rendering Algorithms 21

2.1 Introduction to efficient 3D video rendering 22
  2.1.1 Virtual-view rendering in a 3D video streaming system 22
  2.1.2 Efficient rendering algorithms 22
2.2 Background and related work 24
  2.2.1 3D-scene sampling analysis 25
  2.2.2 Light-field rendering 29
  2.2.3 Depth map and rendering by warping and blending textures 32
  2.2.4 Disparity map and rendering of rectified textures 36
  2.2.5 Summary of virtual-view rendering 39
2.3 Region-based all-in-focus light-field rendering 40
CONTENTS

2.4 Conclusions .................................................. 50

3 Layered Framework for Heterogeneous 3D Video Streaming 53
  3.1 Heterogeneity challenges in 3D video streaming .............. 54
  3.2 Related work ................................................. 56
  3.3 Unifying framework for 3D video streaming .................. 58
    3.3.1 Layered streaming ....................................... 59
    3.3.2 Selective view streaming ................................ 59
    3.3.3 3D data-representation selection ....................... 60
    3.3.4 System architecture proposal .......................... 61
  3.4 Streaming prototype ......................................... 63
    3.4.1 Sender end: layer-reconstruction and compression ........ 65
    3.4.2 Selective streaming with user interaction via a feedback channel .................................. 66
    3.4.3 Multi-stream transmission ................................ 67
    3.4.4 Receiver end: decoding, view rendering and display adaptation ................................ 73
    3.4.5 Experimental results .................................... 76
  3.5 Conclusions .................................................. 82

4 Bandwidth-Adaptive 3D Video Streaming 87
  4.1 Challenges in efficient 3D video streaming over best-effort Internet 88
  4.2 Background and related work ................................ 89
    4.2.1 Best-effort model and fair-rate bandwidth allocation .... 90
    4.2.2 Transport-protocol implementation of rate fairness .... 90
    4.2.3 Conservative streaming rate and continuous buffering .... 92
    4.2.4 Existing work on adaptive 2D video streaming .......... 93
    4.2.5 Summary .................................................. 94
    4.2.6 Novel aspects in adaptive 3D video streaming .......... 95
  4.3 Proposed algorithm .......................................... 95
    4.3.1 Conceptual description ................................... 96
    4.3.2 Spatial virtual-view quality optimization ............... 98
    4.3.3 Spatial and temporal virtual-view quality optimization .. 103
  4.4 Implementation and performance evaluation .................... 105
    4.4.1 Algorithm implementation ................................ 106
    4.4.2 Performance evaluation ................................. 110
5 Interactive Low-Latency 3D Video Streaming

5.1 User interaction and scene navigation in 3D video streaming

5.2 Background and related work

5.2.1 Challenges in continuous 3D video scene-navigation

5.2.2 Related work

5.2.3 Summary

5.3 Proposed algorithm

5.3.1 Conceptual description

5.3.2 User-interaction model for multiview 3D video navigation

5.3.3 User-adaptive video prefetching

5.3.4 Complete algorithm

5.4 Implementation and performance evaluation

5.4.1 Algorithm implementation

5.4.2 Performance evaluation

5.5 Conclusions

6 3D Video Streaming with Remote Rendering

6.1 Large-scale 3D video delivery-challenge

6.2 3D video in conventional delivery architectures

6.2.1 State-of-the-art streaming-CDNs

6.2.2 Analysis of bandwidth cost for providing a 3D video streaming service

6.3 3D video streaming as a distributed rendering application

6.4 Proposed system architecture

6.4.1 3D video streaming with remote rendering

6.4.2 3D server

6.4.3 3D streaming-CDN

6.4.4 Usefulness of a 3D streaming-CDN

6.5 Prototype implementation of remote view rendering and streaming

6.5.1 Remote rendering, coding and streaming at the server end

6.5.2 Decoding and display at the receiver end

6.5.3 Results of the small-scale prototype implementation

6.5.4 Implementation scenarios for a large-scale deployment

6.6 Conclusions

4.5 Conclusions
CONTENTS

7 Conclusions 185
  7.1 Summary and conclusions .................................. 185
  7.2 Discussion on research questions .......................... 187
  7.3 Future work ................................................. 191

Bibliography 193

Curriculum Vitae 215

Acknowledgments 217
Chapter 1

Introduction

1.1 Preliminaries on 3D video

Although 3D still-image viewing already exists since the first half of the twentieth century, the deployment of 3D video systems and television has taken place only recently. 3D-TV based on stereo vision has been gradually introduced in the past five years and this introduction was enabled by the availability of the first generation of 3D-TV displays and the storage of stereo-formatted movies on BluRay disc. In parallel to this development, the development in 3D camera technology has been intense with a rapid succession of various generations. Nowadays, 3D camera systems exist for various principles and are commercially available, like professional stereo cameras, laser/radar-guided depth-sensing cameras and even time-of-flight cameras. These developments support and stimulate the further usage of 3D video systems and as always, the professional applications of this technology will further establish the initial but slowly growing consumer market. Examples of such professional applications that are being deployed are 3D-city modeling and 3D geo-referenced imaging, 3D human-body and organ modeling in medical systems and 3D-film production.

3D video allows a user to perceive depth in the viewed moving scene and to display this moving scene from multiple viewpoints. Stereoscopic video is a stereo video signal with a left and right signal components that can be stored and transmitted as a composed format, featuring also the synchronization of those two signals. Stereoscopic video display is a special case of 3D video viewing, where the scene depth is rendered with the help of a specialized display device.
1. Introduction

– head-mounted glasses or an autostereoscopic display. With *multiple-perspective* video [1], or equivalently *multiview video*, a scene can be displayed from different viewpoints or directions (angles). As a consequence, the format of a multiview video is therefore based on several viewpoints. As an extension, the multiple-perspective viewing can be *interactive*, where the user selects a new viewpoint, or *automatic*, where the user’s movements are continuously tracked and the displayed content is adjusted accordingly. These viewing scenarios are typical for a single-user 3D video system. Alternatively, multiple viewpoints can be rendered simultaneously, if multiple users are present and each has his own renderer. For brevity, we refer jointly to the multiview and stereoscopic video as *3D video* and make clear distinctions where appropriate.

The application domains for 3D video include, amongst others, entertainment, communications, medicine, security, visualization and education. In each of these domains, 3D video brings specific advantages compared to conventional 2D video. In general, these advantages include a sense of immersion, realism of the presentation and particularly for entertainment, enhanced viewing experience. For example, tele-presentation services augment multi-party Internet conferencing with high-quality video rendering [2]. Projection-based video systems create realistic renderings of remote natural scenes and employ autostereoscopic displays to visualize scene depth [3]. Free-Viewpoint Video (FVV) uses multi-camera systems to visualize the scene from arbitrary viewpoints [4]. Distributed collaboration and virtual-reality services enhance productivity, or a sense of immersion, by visualizing large amounts of data in real-time [5].

For the future deployment of 3D video systems in the entertainment domain, the 3DAV working group of the MPEG standard committee has recently studied two application scenarios for in-depth development [6]: Three-Dimensional Television (3D-TV) and Free-Viewpoint Television (FTV). In 3D-TV applications, two closely-spaced images of the same scene are broadcasted simultaneously to create the effect of depth, hence providing a stereo format. In this way, 3D-TV is extending stereoscopic video from the service perspective, by defining a suitable infrastructure for broadcasting such content to the users. With FTV, a scene is broadcasted from several viewpoints, as in the multiple-perspective viewing scenario.
1.2 3D video communication system

A high-level view of a 3D video communication system and its main components is shown in Figure 1.1.

- 3D video is recorded using a multi-camera capturing system. In such a system, a large number of cameras are used to synchronously capture a scene from multiple viewpoints. This provides a basic support for multiple-perspective viewing of the captured scene.

- Scene modelling for 3D video refers to a range of algorithmic approaches for reconstructing a digital representation of a physical scene. These include representation of scene geometry (3D-surfaces), surface reflectance properties and modeling of light sources [7].

- The so-obtained 3D video data-representation consisting of captured video streams and a scene model is compressed before transmission. In this way, the data-rate of the resulting 3D video representation is reduced to enable its transmission over bandwidth-limited communication channels.

- The coded 3D video data can be transmitted to a single receiver or to a number of receivers simultaneously. This includes a transmission over lossy channels with time-varying bandwidth and delay, such as IP-based networks and their wireless extensions, terrestrial and satellite broadcast channels.

- The received 3D video data is decompressed and rendered according to the user’s viewing parameters (view direction or viewing angle). This may include a rendering of scene depth, if a stereoscopic display device is available.

Figure 1.1: 3D video communication system.
1.3 Research scope and assumptions

Stereoscopic and multiview video require new approaches for content capturing, coding, transmission and display [8]. Having discussed the concept in the previous section, we will now define the scope of our research and fundamental assumptions.

In this thesis, we explore the research space for 3D-TV and FTV services, while focusing on 3D video transmission and delivery aspects. The main motivation in doing so is that transmission challenges currently receive little attention in 3D video research. The growth and the generic nature of Internet as a communication system motivate us to develop our 3D video communication system for IP-based networks. In our vision, IP-based networks are best positioned to serve as a substrate for the gradual deployment of 3D-TV and FTV streaming services, and also as their long-term operational environment. The use of IP-based networks is motivated by several arguments. First, the Internet is where the interactive applications have their natural place and the bidirectional nature of the communication allows interactivity to be implemented easily. Second, service deployment over the Internet opens access to a large client base, including a growing number of mobile users, which is important for the service acceptance. Third, Internet endpoints are equipped with programmable processors, and algorithms implementing new functionalities can be realized in software, thereby providing flexibility in experimenting with 3D multiview video formats for different groups of users. Fourth, IP-based networks are increasingly becoming the cornerstone for broadcasting services, such as HDTV. This development makes an extension to 3D video over IP-based networks a natural step.

Common to 3D-TV and FTV services is a requirement for high-bandwidth and controlled-latency transport of large amounts of data between the video capturing and display endpoints. As the service should be available to all endpoints with Internet connectivity, the transport layer should not impose restrictions on their location. We assume that such a transport will have to be realized over a best-effort Internet architecture without affecting net neutrality. This also involves service deployment over bandwidth-limited and error-prone wireless links. The latency aspect will be discussed further in this thesis.

Importantly, the 3D video transport will have to use the network, sender and receiver resources efficiently. Namely, we assume that provisioning of network bandwidth and the involved consumer service will continue to be expensive and a cost-dominating factor when using bandwidth over time. We also assume that the
resources at endpoints, such as access bandwidth, will continue to be scarce and heterogeneous. The resulting requirement for the transport layer can be stated as: finding a network path with sufficiently large bandwidth and low latency between arbitrary endpoints and efficient utilization of the resources available on this path. We conjecture that such a transport and the discussed requirement are essential for the feasibility of new 3D video services in the future Internet.

We assume that stereoscopic cameras are widely available to enable content capturing for 3D-TV. Likewise, the emergence of FTV applications has spawned research interest in building large multi-camera capturing systems. Although such camera systems are not widely available yet, we assume they can be constructed, as exemplified in recent work on this topic (e.g., [9, 4, 10] and the references therein). We also assume that stereoscopic display devices will be widely available to support single-user and multi-user viewing scenarios. This is exemplified by the wide availability of single-view autostereoscopic displays and the increasing quality of the prototypes of multiview autostereoscopic displays [11], respectively.

1.4 Overview of 3D video system technologies

The research space for resource-efficient 3D video transmission systems can be best described with respect to these dimensions: (1) efficient 3D video data-representations, (2) efficient 3D video compression techniques and (3) efficient video streaming algorithms and delivery architectures. The 3D video representations and compression techniques are subject of research in the fields of image-based rendering and video coding, respectively. However, their use in resource-efficient 3D video streaming algorithms and systems has been limited. Similarly, video streaming algorithms and delivery architectures are independently studied in networking and distributed-systems communities. Despite the recent worldwide growth of commercial video streaming services, real-world deployments of 3D-TV streaming algorithms and architectures are limited today. FTV services are deemed a future technology that has seen no deployment up to this date. Despite this slow acceptance, various explorations have been conducted for individual components of a 3D video system, which will be further discussed below.
1. Introduction

1.4.1 3D video data-representations

A distinct new capability of future 3D video systems compared to conventional, 2D video systems is continuous 3D-scene viewing from a user-selected viewpoint. When multiple synchronized cameras are used to record a scene, viewpoint changes can be directly supported by switching to the desired camera stream. However, a 3D video system should also support the more demanding scenario where user-selected viewpoints do not coincide with any of the original cameras.

This capability requires a real-time transmission and processing of a potentially large number of camera streams. To support this capability, the resource demand is significant. These resources include transmission bandwidth, system memory and computation power. An ideal 3D video system that supports scene-viewing from an arbitrary perspective has a resource demand exponentially larger than today’s 2D video systems [4]. Thus, to support continuous viewing from an arbitrary viewpoint, the amount of transmission and computation resources may grow impractically large.

Practical 3D video systems control their resource usage by: (1) constraining the viewing range to a predefined spatial region, (2) reducing the number of capturing cameras and compensating for this reduction by rendering synthetic views from user-selected viewpoints. The first approach is application-specific and requires to design the multi-camera capturing system according to predefined viewing scenarios [9]. The second approach is algorithmic and includes the design of efficient 3D video data-representations and rendering algorithms [12]. Both approaches are guided by theoretical analysis and methodology developed in the field of image-based rendering [12, 13]. Plenoptic-sampling theory models the capturing of a 3D scene with multiple cameras as a process of sampling the plenoptic function [14]. Correspondingly, the multiple-perspective viewing is modeled as a reconstruction of the plenoptic function from the acquired samples.

Virtual-view interpolation, or equivalently virtual-view rendering, in the context of 3D video refers to a set of algorithms that render virtual views of the scene by blending a number of original views (Fig. 1.2). To ensure a seamless transition and a similar image quality across all views, virtual views are automatically constructed from a number of selected original camera streams. These views are typically rendered at locations between and around the original viewpoints [12]. The rendering algorithms either synthesize virtual views directly, or assume some form of scene description (e.g., a geometric model) in order to generate those
views more efficiently [8].

Among different scene models mentioned in Section 1.2, we consider scene-geometry representations only, as our focus is multiple-perspective viewing under original, static lighting. Depending on the scope of geometric information reconstructed for a scene, these representations can be broadly classified as local or global [12]. Global representations reconstruct a geometric model consistent with all input cameras and continuously update it over time (e.g., dynamic 3D-wireframe mesh models). Local representations only describe the scene from a single viewpoint or a subset of viewpoints. One such representation is a depth map. For each pixel in a video frame, a depth map conveys the distance between the camera plane and the nearest surface point in the scene, in the metric coordinate system. A closely related local representation is a disparity map. For each unoccluded pixel in two video frames corresponding to two views of the same scene, the disparity map conveys the distance (offset) between the pixel’s positions, in an image coordinate system.

A complete 3D video data-representation thus consists of a set of original views (commonly referred to as textures) and a representation of the scene-geometry model.

The quality of virtual views rendered for a 3D scene depends on several factors: (1) selection of a suitable 3D video representation and the quality of its
reconstruction, (2) number and selection of textures that are blended together, (3) quality of the employed rendering algorithm and (4) scene complexity. In this context, a resource-efficient 3D video streaming system uses a data representation that maximizes the rendering quality under given resource constraints.

1.4.2 3D video compression

The state-of-the-art video coding standards, MPEG-2 [15], MPEG-4 [16] and H.264/MPEG-4 AVC [17] – although not specifically designed for the compression of 3D video data – are readily applicable to encode a number of 3D video data-representations. By treating each texture as a conventional 2D video, a standard coding algorithm can be applied to each camera stream independently. The same holds for a number of scene-geometry models that have image-like representations, in particular depth and disparity maps.

To account for the specific characteristics of 3D video, a model of human-vision system can be used in addition, to guide the encoding in different qualities or spatio-temporal resolutions. This approach may bring additional bandwidth savings, as demonstrated for stereoscopic video coding [18, 19].

The newly developed Multiview Video Coding standard (MVC) [20] specifically targets efficient compression of multi-camera video recordings. It is largely based on the existing H.264/MPEG-4 AVC standard and extends it with encoding tools optimized for inter-view compression. As the MVC coding algorithm is jointly applied to a number of camera streams, in many scenarios it results in a more efficient compression than an independent H.264/MPEG-4 AVC coding system. The largest compression gains are demonstrated for scenes captured with closely-spaced cameras, where inter-view redundancy is large [21]. This makes MVC particularly attractive for the today’s dominant 3D video type – stereoscopic video – where the camera distance is relatively small. As the result, MVC is expected to be adopted in related content-distribution standards (e.g., BluRay, Digital Video Broadcast (DVB)). However, experiments show that the average bitrate of an MVC-compressed 3D video is still linearly proportional to the number of streams, similar to H.264/MPEG-4 AVC [22]. Further, the MVC does not provide specific encoding tools for scene-geometry models, since a definition of these is out of scope of the standard. However, MVC can be readily applied to encode scene-geometry models for which H.264/MPEG-4 AVC is also applicable. This also holds for coding optimizations based on modeling the human-vision system.
The quality of the virtual views rendered from decompressed 3D video data depends on the following factors: (1) the compression efficiency of the employed coding standard, (2) the average coding rate, (3) the selection of the operational Rate-Distortion (R-D) point and (4) scene complexity. Thus, in the context of 3D video compression, a resource-efficient 3D video streaming system uses a compression configuration that maximizes the rendering quality under given resource constraints.

1.4.3 Video streaming algorithms and delivery architectures

Two basic state-of-the-art network architectures for streaming video are unicast and multicast. In the unicast architecture, a separate copy of a video stream is transmitted over the network to each interested receiver. The network- and server-bandwidth requirements for unicast are thus linearly proportional to the number of receivers. A multicast architecture is conceptually more bandwidth-efficient, as it uses IP-multicast routing algorithms to transport a single copy of the video stream to a group of interested receivers. The network- and server-bandwidth requirements for multicast are linearly proportional to the number of receiver groups. However, due to operational complexity of multicast networks [23], the extent of multicast deployment in the public Internet remains small [24]. Today, multicast-streaming architectures are confined to privately-managed networks and can be found in IP networks that provide IPTV services [25]. In contrast, the unicast architecture is used to deliver roughly 90% of Internet video streaming today [26], despite its inefficient use of bandwidth resources.

The feasibility of unicast streaming in today’s Internet can be largely explained by the bandwidth-overprovisioning practice. The bandwidth overprovisioning is a capacity-provisioning policy where the provisioned bandwidth is several times larger than the expected demand. This practice is common in the capacity dimensioning of today’s networks and servers. To explain how the overprovisioning arises in an end-to-end streaming system, we give examples of current practices in the core networks, the server system and the access networks. Due to overprovisioning in the core networks, it is estimated that the Internet backbone-networks have an average bandwidth utilization of 30-40% [27]. Thus, despite an inefficient use of the core-network bandwidth, unicast streaming traffic is not bottlenecked in the Internet core-networks. The server-bandwidth bottleneck of a unicast streaming architecture has been addressed from the distributed-systems perspec-
tive by deploying streaming-Content Delivery Networks (streaming-CDN) [28, 29]. A streaming-CDN is an overlay network of servers that provide efficient video caching, replication and streaming to receivers, thus reducing the bandwidth requirement on the origin server. The aggregate bandwidth capacity of such distributed-server systems, combined with efficient caching, allows streaming-CDNs to support millions of users [30]. The effectiveness of the combined over-provisioning of server resources and Internet core-networks has even given rise to opportunistic streaming algorithms where the video data are transmitted in short bursts at a rate several times larger than the video coding rate [31, 32, 33]. As a result, the most likely bandwidth bottlenecks for streaming services today are access networks of receivers, including wired [34] and wireless access links [35]. In general, the access networks exhibit lower levels of overprovisioning, compared to core networks and thus cannot be considered grossly overprovisioned [34]. Still, top-of-the-line streaming offerings in the Internet come at a resolution of 1280x720 pixels, frame rate of 25 fps and are encoded at rates of 4–6 Mb/s [36, 37], which makes them accessible to a large number of broadband users [38]. Thus, for a large fraction of today’s Internet low-quality video content, access networks can effectively be considered overprovisioned. However, the heterogeneity of broadband-access bandwidths suggests that high-quality video is still out of reach for a significant fraction of potential users [38].

Although the bandwidth-overprovisioning practice ensures that the bandwidth is sufficient on average, transient drops in available bandwidth often lead to streaming interruptions [31]. As a potential solution, streaming service providers begin to experiment with adaptive streaming, a streaming strategy where the system reacts to changes in the available bandwidth by adapting the streaming rate [39]. A related early standardization activity is Dynamic Adaptive HTTP Streaming (DASH) [40, 41], which focuses on standardizing the exchange of metadata and control signaling in streaming systems. In case of further standardization, DASH may become the first international standard for streaming control.

The video-streaming quality depends on the following factors: (1) the average quality of the rendered video during a session, (2) the number of playout interruptions, (3) the startup delay after which the playout starts and (4) effectiveness of the streaming algorithm. Thus, in the context of 3D video streaming, a resource-efficient 3D video streaming system uses a streaming architecture and a streaming algorithm that maximize the video streaming quality under given resource constraints.
1.5 Problem statement and research questions

A resource-efficient 3D video streaming system should adapt to, or modulate any of the factors in Section 1.4 individually, or combined, to achieve the desired operating point in the quality space within given resource constraints.

The existing research has only partly addressed this problem area.

- Prior research in image-based rendering has indirectly addressed the efficiency of bandwidth utilization for a number of 3D video representations. The focus of image-based rendering research is to achieve efficient rendering of complex scenes based on the assumption that accurate scene models are available. It is generally accepted that the more accurately the selected data representation models a scene, the less data is required to render virtual views [12]. This notion is also formally proved for a class of local scene models [14]. However, the complexity of reconstructing and rendering accurate models of real-world scenes often limits the applicability of such models in real-time 3D video streaming. Further, the research on the applicability of these techniques when used with compressed data has been limited. The direction we want to take in this area is to exploit existing scene models and combine that with the use of compressed data, where compression is based on well-known MPEG-based compression standards.

- Prior research in the area of video compression that led to the recent MVC standard directly addresses the bandwidth efficiency of 3D video coding for storage and streaming. However, the scope of MVC is limited to a definition of the coding tools and the decoder operation and illustrates a strong bias towards existing standards (H.264/MPEG-4 AVC). A specification of an efficient 3D video data-representation and view rendering is out of MVC scope. Thus, most existing research on MVC uses original streams as a 3D video data-representation directly, does not employ scene-geometry models and assumes that the original streams are displayed without virtual-view rendering. We propose that a complete 3D video streaming system should: (1) jointly specify 3D video representation and compression algorithms and (2) adapt those algorithms to specific requirements and challenges in the environment where they are deployed. Our research in this thesis is within the previous statement, but does not address optimization of the specific compression algorithms. Since this area is relatively new, we explore the
two previously mentioned aspects jointly in an experimental setup to realize the first system knowledge in this area.

- Prior research in the area of video streaming has shown that a large-scale delivery of streaming video in the current Internet is possible with over-provisioning at various tiers in the end-to-end system – at the server, in the core and in the access networks. However, there is little evidence that the current video delivery architectures will be able to support future 3D video streaming services in an efficient and cost-effective manner. Despite its simplicity, bandwidth overprovisioning must be regarded as an expensive capacity-dimensioning practice. Although this trade-off may be acceptable for low-quality Internet video today, a similar trade-off may be overly expensive for future high-quality 3D video. First, a single 3D video streaming session requires to transmit a potentially large number of texture and geometry streams. Second, it may require additional bandwidth or processing resources for a low-latency interactive viewpoint adaptation. Third, opportunistic streaming algorithms [31, 32, 33] are unlikely to remain a cost-effective solution under the new requirements. Fourth, although adaptive streaming promises efficient use of the bandwidth resources, efficient adaptive streaming algorithms are still at their infancy [36]. Early standardization initiatives such as DASH aim at defining unifying protocols for streaming control, while the definition of adaptation algorithms is out of the scope of the standard. At the same time, the research literature on adaptive streaming algorithms is scarce, due to a long-time focus on non-adaptive algorithms in the research community [36]. Importantly, most of that research is limited in its focus on streaming rate adaptation, without explicitly considering adaptive video streaming quality [42, 43]. Finally, as of yet, we are not aware of any published adaptive streaming algorithms focusing on 3D video, which will therefore be explored in this thesis. Additionally, due to these distinct new requirements and limitations of existing solutions, we will need new trade-offs in the design of video delivery architectures and new adaptive streaming algorithms.

The limitations in the existing body of research lead to the following research problems that this thesis addresses.

RQ 1: Can we design a good rendering algorithm that allows to employ a sufficiently accurate, yet efficient 3D video data-representation
in a bandwidth-efficient 3D video streaming system?
By defining a suitable processing algorithm after the 3D video capturing stage to reconstruct a scene-geometry model, we can use this model to reduce the bandwidth requirement during streaming. Alternatively, the bandwidth requirement can be reduced without reconstructing a geometric model if we only use a fraction of the available data and define a suitable processing algorithm to apply at the rendering stage. The challenge in both cases is to achieve bandwidth efficiency without compromising the quality of rendered virtual views. Our thesis splits this research question in two specific questions: (1) we explore depth- and disparity-based models and their use in a streaming system, (2) we explore alternative models in the form of a light-field representation and propose a rendering algorithm for this representation.

RQ 2: How can we provide a unifying solution to heterogeneity in 3D video streaming systems?
An important requirement for a 3D video streaming system is to simultaneously accommodate receivers that have highly heterogeneous resources such as access bandwidths and display devices. A further requirement is that the view rendering should enhance a sense of immersion, which leads to heterogeneity of viewing preferences in addition. This suggests that the most resource-efficient 3D video data-representation is the one that optimally matches the resource level and preferences of each user. However, finding an optimal representation for a large number of concurrent, heterogeneous users will limit the scalability of real-world system realizations. This raises a problem of jointly defining an efficient system model that accounts for highly heterogeneous users and a 3D data-representation to use with this model. Two specific research questions addressed in this thesis are: (1) definition of a 3D video representation and the corresponding streaming model that serves heterogeneous users, (2) definition of a streaming architecture that allows to support even resource-impoverished users.

RQ 3: Can we design algorithms for bandwidth-efficient and low-latency 3D video streaming over best-effort networks that can handle the time-varying available bandwidth and delay?
A 3D video streaming service is characterized by simultaneous requirements for: (1) large bandwidth and (2) low latency. The first challenge is that
bandwidth is still a limited and expensive network service. It is also time-varying, as it is provided in a best-effort fashion. The second challenge is that end-to-end system delays may be large and are rarely optimized for in the design of network systems and architectures – the system delay is typically compromised for other important system functionalities like security and load balancing. A resource-efficient 3D video streaming system should employ streaming algorithms that adapt to these conditions while maximizing the rendered quality. This raises the question of the design of adaptive algorithms that jointly optimize the underlying 3D video representation and compression algorithms while maximizing video quality under dynamic bandwidth conditions.

To the best of our knowledge, this thesis is the first to address the above research questions in an integrated fashion for 3D video, as well as the first to propose algorithmic solutions for adaptive 3D video streaming.

1.6 Research contributions

The research presented in this thesis focuses on algorithms for resource-efficient and resource-adaptive 3D video streaming. We also consider system architecture, analysis, simulation and build a 3D video streaming prototype. The thesis contributions are as follows:

Efficient 3D video representations and algorithms for continuous viewpoint navigation using virtual-view rendering (Chapter 2).

We present three 3D video representations and design algorithms that allow for efficient continuous rendering of virtual views in the following cases.

1. Texture, depth map and an algorithm for view interpolation using depth-based warping and blending.
2. Texture, disparity map and occlusion map (defining inter-camera occlusion relationships) and an algorithm for blending of rectified textures.

The described techniques are, generally speaking, independent from reducing the bandwidth requirement through video compression and are complementary to it. The first and second algorithms are not original algorithmic
contributions of this thesis, but the contribution lies in adapting them to build a streaming prototype and to develop efficient streaming algorithms, respectively. The third algorithm is an original algorithmic contribution of this thesis and has been developed jointly with Aneez Kadermohideen Shahulhameed during his M.Sc. thesis project [44, 45]. Related international publications describing the above results can be found in [46], [44], [47] and [48].

Efficient framework for heterogeneous 3D video streaming and system architecture for large-scale 3D video delivery (Chapter 3 and Chapter 6).

In Chapter 3, we propose a unified streaming framework that addresses the heterogeneity problem without compromising scalability and build a streaming prototype according to this framework. Our prototype is acknowledged as one of first two stereoscopic-streaming prototypes in the research community [49]. The system adopts a local 3D video representation in the form of texture and depth or disparity map and demonstrates the following principles.

1. **Layered streaming** for quality scalability. The number of layers to receive for a single virtual view can be matched to the average available bandwidth, the type of a display device or user’s viewing preference.

2. **Selective view streaming** with real-time user-navigation feedback for view scalability. The number of views to receive can be interactively matched to the average available bandwidth or viewing preferences.

3. **Remote view rendering.** The focus of Chapter 6 is to show that by regarding the 3D video streaming system as a distributed application and offloading the view-rendering functionality to a remote location, we can significantly reduce the bandwidth requirements and thus support even resource-impoverished receivers.

The research results on the above aspects are covered in the following publications: [46], [50], [51] and [52].

Adaptive algorithms for interactive 3D video streaming (Chapter 4 and Chapter 5).

This contribution consists of two major lines of work.
1. We propose an algorithm that achieves a continuous streaming and high-quality multiview rendering over best-effort networks, despite the time-varying available bandwidth. The algorithm performs a virtual-view streaming adaptation using an optimized joint texture-depth rate allocation. This algorithm is acknowledged in the community as the first algorithm for adaptive 3D video streaming that performs a joint optimization of the 3D data-representation, its rendering algorithm and the compression algorithm [53].

2. We propose an algorithm that achieves a bandwidth-efficient and low-latency interactive 3D-scene navigation in streaming systems with large and time-varying delay. The algorithm employs user-adaptive video prefetching to reduce the perceived interaction latency and performs a quality-optimized rate allocation of the prefetching data. To the best of our knowledge, this is the first streaming algorithm that is designed to adapt to both the available bandwidth and the user-interaction patterns.

We evaluate both algorithms using a methodology that combines network simulations with video coding and rendering experiments. Related publications are [54], [55], [56] and [57].

1.7 Thesis outline and publication history

The research work and the main contributions of this thesis are already published. This thesis synthesizes the work and is structured as follows. In Figure 1.3, we provide a roadmap to the thesis in order to help the reader quickly locate our main contributions and place each chapter’s content into the problem space given in Section 1.4. A more detailed overview of each chapter’s content and the related international publications covering the obtained results are provided next.

Chapter 2 focuses on continuous 3D-scene viewing from multiple viewpoints as a distinct new capability of future 3D video systems compared to conventional, 2D video systems. To support this capability, the resource demand is significant. We start by an analysis showing how the bandwidth requirement of a multiview 3D video system depends on scene-sampling rate. In a streaming system, a high scene-sampling rate translates to high and scene-dependent cost of provisioning the network bandwidth. To reduce this cost, a resource-efficient 3D
video streaming system needs to maximize the rendering quality under a given bandwidth constraint. In severely bandwidth-constrained scenarios, rendering artifacts occur in the synthesized virtual views. In this chapter, we first provide an overview of visually-disturbing artifacts that appear when rendering virtual views under limited bandwidth. Subsequently, we provide a high-level description of two rendering algorithms that rely on scene-geometry models in the form of depth and disparity maps, respectively. These algorithms are not our original contributions, but serve as the rendering basis for the algorithmic contributions made in subsequent chapters of this thesis (Chapters 4, 5 and 6). The central part of this chapter is an algorithm for virtual-view rendering using a light-field representation. Specifically, we propose an algorithm for high-quality rendering of undersampled light fields. The proposed algorithm – referred to as region-based all-in-focus light-field rendering – incorporates image segmentation to dynamically adapt the light-field rendering to scene depth-complexity. This algorithm is an original contribution of this chapter and is published in IEEE Int. Conf. on Image Processing (ICIP) 2009 [44].

Chapter 3 addresses the challenge of heterogeneity of resources available for 3D video streaming in the Internet as well as the heterogeneity of viewing preferences. The resource heterogeneity is a challenge for a 3D video delivery system that aims to simultaneously accommodate many users with highly heterogeneous resources. We start this chapter with an overview of heterogeneity challenges in 3D video streaming. Next, we propose a layered framework for 3D video streaming as a unifying and efficient solution to the problem of heterogeneity of resources and viewing preferences. The architecture components of our framework include: (1) efficient 3D video representation, (2) efficient decomposition of the 3D-scene description into information layers, where each layer conveys a single coded video signal or coded scene-geometry data, and (3) rendering of virtual views. Heterogeneous receivers can select the number of layers to receive for view rendering, depending on the availability of resources or viewing preferences. To demonstrate the viability of the proposed architecture, we implement a 3D video streaming prototype and show that heterogeneous autostereoscopic 3D displays can be supported by the system. Our layered framework is first published in IEEE Int. Conf. on Multimedia and Expo (ICME) 2006 [47] and Benelux Information Theory Symposium (WIC) 2006 [57]. The complete system implementation is presented at SPIE Electronic Imaging 2008 [46] and Benelux Information Theory Symposium (WIC) 2007 [50]. Additional application scenarios are published in Int. Conf.
on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine) 2007 [52] and IEEE Int. Conf. on Global Communications (Globecom) workshop 2008 [56].

Chapter 4 focuses on the quality of 3D video streaming in transmission scenarios characterized by a varying availability of bandwidth and computation resources in general and network bandwidth in particular. Due to statistical bandwidth sharing in the Internet, the available bandwidth varies over time and location due to competing traffic on shared links. Transient drops in available bandwidth lead to streaming interruptions that negatively effect immersiveness of 3D video streaming. Correspondingly, we propose an algorithm for bandwidth-efficient 3D video streaming that achieves a continuous streaming and a high rendering qual-
ity, despite the variations of available bandwidth. The main contribution in this chapter is to demonstrate that quality-optimized 3D video streaming algorithms should: (1) adapt the streaming rate by explicitly considering rendering quality, and (2) employ a 3D video adaptation that jointly considers the constituent components of the 3D video data-representation, the coding algorithm and the rendering algorithm. The evaluation results demonstrate a significantly higher average video quality over quality-agnostic and conventional streaming strategies. This algorithm is published in SPIE Electronic Imaging 2010 [54].

In Chapter 5, we propose a 3D video streaming algorithm that achieves a low interaction latency and high rendering quality, even in streaming systems with a large and time-varying system delay. Our main contribution in this chapter is to demonstrate that future 3D video streaming systems should employ streaming algorithms that: (1) explicitly minimize user-perceived latency, (2) adapt to navigation patterns of the user and the available bandwidth and (3) explicitly control rendering quality during 3D-scene navigation. The proposed algorithm prefetches texture and depth streams in order to reduce the perceived latency, adapts the 3D video streaming rate – including the prefetching rate – to increase its bandwidth efficiency and minimizes quality variations among consecutively-rendered views of the 3D scene. As a result, it ensures a visually-smooth navigation and a sense of immersion. This algorithm is partly published in the Picture Coding Symposium (PCS) workshop 2010 [55] and extends the algorithm proposed in Chapter 4.

Chapter 6 complements the previous chapters by focusing on efficient architectures for the delivery of 3D video streams to a large number of concurrent users. In this chapter, we first review current solutions for large-scale deployment of Internet streaming services and present an analysis of their cost structure. We then motivate a fresh view on 3D video delivery architectures and cost optimizations. Specifically, we argue that the architectural view of a 3D video streaming system as an application with distributed data and distributed computation is an appropriate model for efficiency optimizations in a large system with resource-constrained and heterogeneous users. The main contribution of this chapter is a streaming architecture that consists of the following components: (1) streaming-CDN, (2) rendering of virtual views provided as a service of the streaming-CDN and (3) real-time in-CDN encoding and streaming. We describe our implementation of a 3D video streaming prototype according to the proposed architecture and experimentally demonstrate its ability to efficiently support resource-constrained receivers. We also provide an overview of the technology trends that allow for
efficient large-scale implementations of this architecture. The work in this chapter is published in Int. Conf. on Immersive Telecommunications (Immerscom) 2007 [51].

In Chapter 7, we conclude the thesis, present our solutions to the research problems stated in Section 1.5 and the insights obtained through analysis, simulation, emulation and prototype building. At the end of the chapter, we also propose several directions for future work based on the findings of this thesis.
Chapter 2

Efficient 3D Video Representations and Rendering Algorithms

In this chapter, we present three virtual-view rendering algorithms that employ efficient 3D video representations to significantly reduce the bandwidth requirement of a 3D video streaming system at a negligible penalty in rendering quality. The first and second algorithms employ 3D video representations that combine textures with scene-geometry models in the form of depth and disparity maps, respectively. They achieve a high rendering quality by relying on principles of multiview geometry of a 3D scene to guide the rendering process. These algorithms are not our original contributions, but they form the rendering basis for streaming algorithms developed in subsequent chapters of this thesis. Next, we propose an algorithm that renders undersampled light-field representations of 3D scenes at a high visual quality. This is achieved by minimizing rendering artifacts caused by aliasing in the synthesized views. The proposed algorithm – referred to as region-based all-in-focus light-field rendering – incorporates image segmentation to dynamically adapt the light-field rendering to scene depth-complexity. As a result, it renders all-in-focus virtual views of a 3D scene at a quality visually indistinguishable from the original views. The algorithm is an original contribution of this chapter, developed jointly with the co-author of an earlier conference publication [44].
2. Efficient 3D Video Representations and Rendering Algorithms

2.1 Introduction to efficient 3D video rendering

Our work in this chapter focuses on efficient 3D video data-representations and virtual-view rendering algorithms. As discussed in Section 1.4, despite the significant progress in the field of image-based rendering, the use of these algorithms in streaming systems has been limited up to the ending period of writing this thesis. In this chapter, we first perform an analysis to identify the main challenges and then propose algorithmic solutions for these challenges.

2.1.1 Virtual-view rendering in a 3D video streaming system

We use an analysis inspired by the plenoptic-sampling theory to show that 3D-scene sampling has important consequences for the feasibility and efficiency of 3D video streaming systems. The primary focus of this analysis are the resource demands of 3D video streaming with virtual-view rendering. Intuitively, the bandwidth requirement of a 3D video system depends on the scene-sampling rate. As our analysis will show, the bandwidth required to achieve an accurate reconstruction of the plenoptic function is scene-dependent and may be very large. In a streaming system, this translates to high and scene-dependent costs of provisioning the network bandwidth. Similarly, a reconstruction of the plenoptic function is computationally intensive, due to the requirement to process a large number of samples. This requires a significant computation power at streaming endpoints, which is generally scarce. Both of these factors present significant challenges to deployment of 3D video streaming systems. Summarizing, the magnitude of the required bandwidth is the main limiting factor for the deployment of 3D video streaming systems.

2.1.2 Efficient rendering algorithms

The main contribution of this chapter are virtual-view rendering algorithms that address the above challenges and allow to significantly reduce the resource demands, thus enabling more efficient 3D video streaming systems. Our solution approach is as follows.

Our main observation is that the stated goals of a 3D video streaming system – enhanced immersiveness and realism of presentation – can be achieved if the system attains a similar rendering quality across all rendered views. Given the requirement for multiple-perspective viewing, this means that the virtual views
of a scene should be rendered at a quality comparable to original views. Correspondingly, the main idea in our approach is to reformulate the rendering problem of accurately reconstructing the plenoptic function as a problem of rendering virtual views that are visually indistinguishable from the original views. In turn, this allows to employ undersampled 3D-scene representations and significantly reduce the bandwidth and computational requirements. As a result, a resource-efficient 3D video streaming system can use a data representation and a rendering algorithm, which jointly maximize the rendering quality under given resource constraints.

Although prior research in image-based rendering provides examples of efficient data-representations and algorithms, our work in this chapter shows that they are unable to provide sufficient rendering quality when the scene is undersampled. In particular, state-of-the-art algorithms render undersampled 3D-scene representations with visually-disturbing artifacts. In this chapter, we provide an overview of artifacts typical of widely-used 3D video representations and of the associated rendering challenges.

Rendering algorithms that resolve these challenges are the main focus and contribution of this chapter. By defining a suitable processing algorithm after the 3D video capturing stage to reconstruct a scene-geometry model, we can use this model to reduce the bandwidth requirement during streaming. Alternatively, the bandwidth requirement can be reduced without reconstructing a geometric model, if we only use a fraction of the available data and define a suitable processing algorithm to apply at the rendering stage. The algorithmic challenge in both cases is to achieve bandwidth efficiency without compromising the quality of rendered virtual views. In the scope of this thesis, we introduce the term efficient rendering algorithm to mean an algorithm that achieves at least a comparable performance (in terms of the rendering quality) as the best algorithm of the day, when rendering virtual views of the given scene, while using the same original views and the same geometry information. The essential in this definition is that it refers to bandwidth-efficiency of the rendering process, because this is one of the primary performance parameters of rendering algorithms.

In Figure 2.1, we provide a roadmap to this chapter in order to help the reader quickly locate our main contribution and place this chapter’s content into the problem space given in Section 1.4. A more detailed overview is as follows. We use plenoptic-sampling analysis for 3D scenes and provide an overview of visually-disturbing artifacts that appear when rendering virtual views of undersampled 3D
scenarios in Section 2.2.1. This is followed by a discussion of related algorithmic work on rendering for three common 3D data-representations: a purely image-based light-field representation (Section 2.2.2) and two representations that additionally rely on scene-geometry models in the form of depth (Section 2.2.3) and disparity maps (Section 2.2.4), respectively. We note that the algorithms presented in Section 2.2 are not our original contributions, but serve as the rendering-basis for the algorithmic contributions made in subsequent chapters of this thesis. In Section 2.3, we propose an algorithm for high-quality rendering of undersampled light fields. This algorithm is an original contribution of this thesis, jointly developed with Aneez Kadermohideen Shahulhameed [44]. In Section 2.4, we conclude the chapter.

2.2 Background and related work

As stated in Section 2.1, virtual-view rendering algorithms are guided by theoretical analysis and methodology developed in the field of image-based rendering [12, 13]. Most importantly, the plenoptic-sampling theory provides insights into sampling and reconstruction of the plenoptic function from the acquired samples.
2.2.1 3D-scene sampling analysis

A. Plenoptic function and scene sampling

The plenoptic function is a multivariate function that models the intensity of light in free space, first defined in [58]:

\[ P_\gamma(x, y, z, \theta, \phi, \lambda, t), \]  

(2.1)

where \((x, y, z)\) are the spatial coordinates of the intensity-measurement point, \((\theta, \phi)\) represent the incident direction, and \(\lambda\) and \(t\) stand for wavelength and time, respectively. According to this model, the approach of constraining the viewing range in a 3D video system (Section 1.4.1) corresponds to partial sampling and reconstruction of the plenoptic function. In this way, sampling analysis is limited to the spatial region that is in the field-of-view of the employed multi-camera system. The number of cameras in the system, their location and sensor resolution determine the sampling rate in this region. Correspondingly, virtual-view rendering amounts to reconstructing the plenoptic function in this region by interpolation from the available samples.

Based on this theoretical foundation, practical 3D video capturing-systems limit the spatial viewing region and the sampling rate in order to reduce the sys-
tem cost [9]. The multi-camera capturing systems for 3D video are typically constructed as 1D or 2D camera arrays. Figure 2.2(b) illustrates a 2D array consisting of two horizontal rows of cameras, with $N$ cameras in each row. This array limits the spatial viewing region in horizontal direction to the combined field-of-view of cameras $\text{Cam}^{(i,1)}, \ldots, \text{Cam}^{(i,N)}$ and in vertical direction to the field-of-view of the two camera rows $\text{Cam}^{(1,n)}$ and $\text{Cam}^{(2,n)}$, with $i = 1, 2$ and $n = 1, \ldots, N$. Within this region, a virtual frame can be synthesized at an arbitrary position by interpolation from adjacent original cameras. Figure 2.2(b) illustrates two virtual views $V^{(1)}$ and $V^{(2)}$, synthesized at positions $(V_{x}^{(1)}, V_{y}^{(1)})$ and $(V_{x}^{(2)}, V_{y}^{(2)})$, respectively. The 2D coordinates refer to the center of a virtual camera. We omit the $z$ coordinate in this example by assuming a planar camera array such that the original-camera centers $(C_{x}^{(1,1)}, C_{y}^{(1,1)}), (C_{x}^{(2,1)}, C_{y}^{(2,1)}), \ldots, (C_{x}^{(1,N)}, C_{y}^{(1,N)}), (C_{x}^{(2,N)}, C_{y}^{(2,N)})$ and the two virtual-camera centers are in the same plane. This assumption does not limit our analysis and is often satisfied in practice since most existing prototypes of 2D camera arrays are planar [9, 10, 3]. If we further assume that a practical camera sensor takes RGB samples instead of sampling the entire range of wavelengths $\lambda$, the function in Eq. (2.1) can be rewritten as:

$$P_{5}(x, y, \theta, \phi, t).$$

Each pixel value in an original-camera frame represents a sample of the plenoptic function. Correspondingly, each pixel value in a virtual frame represents a reconstruction of the plenoptic function through interpolation of the pixel values in original cameras. In the example in Figure 2.2(b), the synthesis of virtual views $V^{1}$ and $V^{2}$ requires samples from two and four original cameras, respectively.

The maximum sampling rate achievable with a camera array is determined by the resolution of the employed camera sensor and the physical distance between adjacent cameras. For our purposes, we can assume that sensor resolution is the same for all cameras, which is the case in existing prototypes [9, 4, 10]. Camera baselines represent vertical and horizontal distances between the centers of adjacent cameras [13]. For example, the horizontal baseline of cameras $C^{(1,1)}$ and $C^{(1,2)}$ is simply $C_{x}^{(1,2)} - C_{x}^{(1,1)}$ (Fig. 2.2(b)). Camera arrays are typically constructed such that the horizontal and vertical baselines are equal for all pairs of adjacent cameras. Thus, the sampling rate is primarily determined by the array size – the number of cameras in the array, or equivalently – the camera baseline.

In general, appropriate array size is scene-dependent and should be selected
such that the light leaving each 3D-scene point is sampled by at least one camera [12]. In practice, this is ensured by placing the cameras closely enough such that their fields-of-view overlap (Fig. 2.2(b)). However, determining an optimized camera baseline for the given 3D scene is a difficult problem in plenoptic-sampling theory [14]. This problem can be formulated as finding the maximum camera baseline that allows the plenoptic function of the 3D scene to be reconstructed in the spatial region delimited by the array. The first important finding of the analysis in [14] is that the sampling rate required for the given 3D scene is bounded by the minimum and maximum depths of the scene. Equivalently, if the minimum ($Z_{min}$) and maximum ($Z_{max}$) depths are known, the maximum camera baseline can be determined to guarantee accurate reconstruction of the function in Eq. (2.2). As a result, it will be possible to render accurate virtual views of the 3D scene, in the sense of accurately approximating the plenoptic function at each respective viewpoint. The second important finding in [14] is that for such an accurate reconstruction, the disparities in all pairs of adjacent original-camera frames must not exceed one pixel. In practice, the array size required to guarantee the maximum disparity of one pixel is scene-dependent and may be very large [9]. Correspondingly, the bandwidth and computation requirements of a 3D video streaming system to support multiple-perspective viewing at these sampling rates may lead to a very high and potentially impractical system cost.

B. Rendering-quality challenges

A 3D video streaming system must support multiple-perspective viewing at a practical system cost. According to the above sampling analysis, the main challenge is to ensure that virtual views can be rendered at a quality comparable to original views, even if the scene-sampling rate is suboptimal in the sense of accurately reconstructing the plenoptic function. We address this challenge by employing efficient 3D data-representations and rendering algorithms. With this problem formulation, quality of virtual views depends on: (1) the selection of a suitable 3D video representation and the quality of its reconstruction, (2) the number and the selection of original-camera streams – or equivalently, textures – that are blended together, (3) the quality of the employed rendering algorithm and (4) scene complexity.

Prior research in image-based rendering has addressed the efficiency of 3D data-representations and rendering algorithms [12]. However, open issues con-
cerning quality remain with respect to *minimizing rendering artifacts* that result from 3D-scene undersampling. Specifically, the rendering quality may be affected by the following artifacts in synthesized virtual frames.

- **Blurred rendered frame.** This artifact appears due to aliasing in the frequency-domain representation of an undersampled plenoptic function [14]. It can also manifest itself as a double image in specific areas of the frame, or the entire synthesized frame. Visually, the synthesized frame appears out-of-focus. The interpretation of this artifact in the spatial domain is that due to a sparse sampling, the same pixel positions in original cameras correspond to different points in the 3D scene. If those points are at largely different depths, the pixels synthesized by interpolation will have incorrect values. In Section 2.3, we propose a rendering algorithm specifically designed to minimize this artifact.

- **Holes in the frame.** This artifact is common with 3D data-representations that employ a scene-geometry model. As stated in Section 1.3, the availability of such a model allows to render virtual views more efficiently. More formally, the plenoptic-sampling analysis shows that the availability of a model for 3D-scene depth can compensate for a sparser sampling and enable an accurate reconstruction of the plenoptic function [14]. However, the underlying assumption is that a scene-depth model can be reconstructed accurately, precisely and at a practical computation cost. These assumptions are often violated in practice [59, 60, 61]. Due to limited accuracy and precision of practical model reconstructions, virtual frames may contain areas in which the pixel values cannot be synthesized correctly or not at all. In Sections 2.2.3 and 2.2.4, we present rendering algorithms that explicitly address these artifacts.

- **Contour artifacts – ghosting and noise.** High-frequency components of original frames that coincide with depth discontinuities are especially challenging for rendering using geometric models. First, these “contour pixels”, corresponding to contours of image objects, typically contain a mixture of foreground and background colors [62]. In the synthesized views, these pixels often lead to “ghosting” artifacts in the form of double contours of image objects. The algorithm in Section 2.2.3 is designed to explicitly minimize

---

1In [63] and [64], the term “ghosting” denotes the occurrence of double images due to aliasing,
this artifact. Second, an accurate reconstruction of the scene depth for these pixels is usually error-prone. Rendering the contour pixels with inaccurate depth leads to mosquito-noise type of artifacts along the object contours or coarse contours in the synthesized views [62].

- *Artifacts due to scene complexity (glossy and specular surfaces, occlusions).* In general, 3D scenes that contain reflective surfaces or complex occlusions are very challenging for image-based rendering and require high sampling rates [12]. Determining an optimized sampling rate is difficult and such cases are not considered in the plenoptic-sampling analysis in [14]. Although algorithms can be developed to estimate the scene complexity and dynamically increase the sampling rate, they are rarely reported in the literature [12].

### 2.2.2 Light-field rendering

**Light field** is a 3D data-representation first introduced by Levoy and Hanrahan [66]. The light field represents the plenoptic function in Eq. (2.2) as a set of light rays, where each ray is parametrized by its intersection with two planes (Fig. 2.3). The plenoptic function in Eq. (2.2) represented as a light field, can be rewritten as:

\[
P_4(u, v, s, t),
\]

(2.3)

where \((u, v)\) are the coordinates of the intersection of a light ray with the focal plane \((u', v')\) in Fig. 2.3, \((s, t)\) are the coordinates of its intersection with camera plane \((s', t')\) in Fig. 2.3 and the time dimension is omitted for simplicity such that we can assume that one light field is sampled in each discrete time interval \(^2\). In the absence of occluders and light dispersion, the light field is a complete representation of the plenoptic function in the spatial region that is in the field-of-view of the camera array [66] \(^3\).

The second major contribution of the work in [66] is *light-field rendering*, an algorithm that synthesizes virtual views of a 3D scene using the light-field

\(^2\)The notation from [66] uses \((u, v)\) coordinates for the camera plane and \((s, t)\) coordinates for the focal plane. Instead, we follow the notation from [14] and [67].

\(^3\)The light-field representation in [66] should not be confused with the independently proposed ray-space representation in [68].
representation. As in the above plenoptic-sampling analysis, each pixel value in an original camera represents a sample of the light field. This sample corresponds to an average of all light rays leaving the 3D scene and intersecting at that pixel position [66]. The light-field rendering synthesizes virtual views as follows. The coordinates of each ray intersecting the required pixel position in the virtual view are used to select the samples in adjacent original cameras. This is achieved by first computing the intersection of a ray and the focal plane. Next, the algorithm estimates pixels in the original cameras that correspond to this intersection point by back-projecting it to the camera plane. Finally, the pixel value is synthesized by interpolation from the corresponding pixels in the original cameras. Nearest-neighbor, bilinear or quadrilinear interpolation are commonly used to compute final pixel values in the virtual view.

Although the light-field rendering algorithm in [66] demonstrates compelling results, these results are achieved with densely-sampled light fields. The algorithm assumes that the 3D scene is close to the focal plane, which allows to position the focal plane at a particular depth for rendering. Effectively, the choice of depth for the focal plane determines the parts of the scene that will be rendered in focus. However, the scene parts at depths significantly different from the focal-plane depth may appear out-of-focus. As stated in Section 2.2.1 for general 3D data-representations, such out-of-focus rendering will be especially pronounced for undersampled light fields and lead to visually disturbing blur in the synthesized views.

We note that an a-priori decision as to what parts of the scene will be rendered in focus, i.e., without aliasing, may be undesirable [63]. For this reason, several researchers have addressed the general problem of light-field rendering without
aliasing. We survey these approaches in the next section.

In the sequel, we discuss related work in the area of rendering undersampled light fields without aliasing, commonly referred to as all-in-focus light-field rendering.

To suppress aliasing artifacts when rendering virtual views of an undersampled 3D scene, prefiltering of the light field is discussed in [66]. The prefiltering can be implemented algorithmically by first oversampling the scene with a large array and then applying a low-pass filter. Oversampling is often impractical as the resulting sampled data is very large. Moreover, as observed in [63], this approach makes an unfavorable a-priori decision as to what parts of the scene will be rendered in focus. Instead, Isaksen et al. [69] introduce the concept of a movable focal plane. They render scene objects at different depths by dynamically positioning the focal plane. To reduce the aliasing, they increase the aperture of the reconstruction filter used in the interpolation. However, an issue with the so-obtained wide-aperture filter is that this filter removes the high-frequency content and tends to produce blurry renderings in many practical situations [63].

A number of recent proposals extend the ideas in [69] and combine renderings at multiple focal planes to produce an all-in-focus virtual frame. In this approach, different focal planes are used to construct a simple geometric model that consists of a small number of depth layers. The light-field rendering algorithm is then successively applied by positioning the focal plane according to the depth of each layer, and then combining these intermediate frames to render the final virtual frame. The use of image segmentation as a means of estimating the depth layers is considered in [70] and [71]. However, in our algorithm, we will use image segmentation in an alternative way, to enhance the performance of a light-field rendering algorithm. We have recently performed an extensive study to compare these proposals [45]. Most of these techniques are implemented as pre-processing steps and many of them require user input. A notable exception is the algorithm of Takahashi and Naemura [64] that uses spatial consistency of different reconstruction filters to estimate the depth layers during rendering. Since our objective is to perform all-in-focus light-field rendering automatically and without pre-processing, the method in [64] is most closely related to the algorithm we propose in Section 2.3.
2.2.3 Depth map and rendering by warping and blending textures

As stated in Section 2.2.1, the availability of a model for 3D-scene depth can compensate for a sparser sampling and enable efficient virtual-view rendering.

A 3D data-representation that explicitly conveys the scene-geometry information in the form of depth layers is 1-texture/1-depth [72]. For each pixel in a texture frame, the associated depth map contains the distance between the camera plane and the nearest surface point in the 3D scene (Fig. 2.5 (b)). The depth map is typically formatted as a greyscale image whose intensity values \(I_{\text{depth}}(z)\) correspond to scene depth \(z\) as in [22], giving:

\[
I_{\text{depth}}(z) = \text{round}\left[255 \cdot \left(1 - \frac{1}{Z_{\text{max}}}\right) / \left(\frac{1}{Z_{\text{min}}} - \frac{1}{Z_{\text{max}}}\right)\right],
\]

where \(Z_{\text{max}}\) and \(Z_{\text{min}}\) are the maximum and minimum scene depth, respectively. If the scene is captured with a camera array, this representation generalizes to \(N\)-texture/\(N\)-depth [62, 72], where each original camera is associated with a depth map.

Since the depth maps are an explicit component of this 3D data representation, they need to be generated, stored and transmitted. For an overview of algorithms for automatic estimation of depth maps from video frames, the interested reader is referred to a recent survey [8, 59]. Recently, hardware for scene-depth estimation has also become available [73].

With the 1-texture/1-depth representation, multiple virtual views can be rendered by re-projecting the original texture pixels into the new viewpoint, based on the depth map. The class of rendering algorithms that use a depth-based representation is commonly referred to as Depth Image Based Rendering (DIBR) in the literature [12]. More formally, Eq. (2.5) represents a DIBR process in a matrix notation, using homogeneous coordinates:

\[
\lambda \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & \alpha f_x & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & 0_3 \\ 0_3^T & 1 \end{bmatrix} \begin{bmatrix} 1_{3\times3} & -I \\ 0_3 & 1 \end{bmatrix} \begin{pmatrix} x_s \\ y_s \\ z_s \\ 1 \end{pmatrix}.
\]

A full camera calibration is required by the rendering process [72]. The rendering maps each visible 3D-scene point \((x_s, y_s, z_s, 1)^T\) to a point \((x_i, y_i, 1)^T\) in a 2D image space, up to a scaling factor \(\lambda\).
We illustrate this with an example given in Figure 2.4. By first mapping all image space points \((x_i^{(N)}, y_i^{(N)}, 1)\) of a reference camera \(C^{(N)}\) to the scene space according to the inverse of Eq. (2.5), and then mapping the resulting scene points \((x_s^{(N)}, y_s^{(N)}, z_s^{(N)}, 1)\) back into image space according to Eq. (2.5), we generate a virtual view \(V^{(N)}\) at an arbitrary position \((x_i^{V(N)}, y_i^{V(N)}, 1)\), where \(x_i^{V(N)}\) and \(y_i^{V(N)}\) refer to the center of the view \(V^{(N)}\). The details of the rendering process can be found in [72]. For completeness, we note that the Eq. (2.5) models the process of re-projecting the camera \(C^{(N)}\) to a virtual view \(V^{(N)}\) with the following parameters: camera translation \(t\), rotation matrix \(R\) and the physical properties of the employed imaging sensor – focal lengths \((f_x, f_y)\), principal point \((c_x, c_y)\) and skew coefficient \(\alpha\).

In practice, DIBR algorithms are usually implemented as a form of 3D image warping, first proposed by McMillan [74]. A 3D image warping algorithm uses the depth map to directly shift the texture pixels from an original view to a virtual view, while resolving the occlusion relationships [74].

Figure 2.5 illustrates an original texture, its depth map and a virtual rendered frame, using a DIBR algorithm. In general, DIBR using 1-texture/1-depth is effective for rendering virtual views close to an original camera. In this case, the distortions in synthesized virtual views are usually imperceptible. However, as this figure shows, large “dissoclusions” seriously affect the rendering quality for

\[\text{Figure 2.4: Re-projection of an original view corresponding to camera } C^{(N)} \text{ to a virtual view } V^{(N)}.\]

\[\text{Figure 2.5: An original texture, its depth map and a virtual rendered frame using a DIBR algorithm.}\]

\[\text{4The “Ballet” multiview sequence is kindly provided by Microsoft Corporation [62].}\]
Since no general algorithmic solution exists to the problem of large disocclusions, practical rendering algorithms combine samples from multiple original cameras. This is accomplished using the N-texture/N-depth representation. If the scene-sampling rate is sufficiently high, scene parts occluded in one camera will be visible in one of the neighboring cameras. The rendering algorithm then blends the frames from a selected set of cameras, while taking into account the visibility relationships between them [62].

In a number of streaming experiments in this thesis, we employ the view-rendering algorithm for the N-texture/N-depth representation proposed in [65]. This algorithm enables a continuous horizontal blending between pairs of original cameras. For each pair of adjacent cameras, the input to the algorithm includes: (1) left texture; (2) right texture; (3) left depth map; (4) right depth map. The algorithm creates a virtual view at an arbitrary position between the cameras. The rendering consists of the following steps.

**Depth-based rendering**

**Step 1: Warping.**
Both the texture and the depth of the selected two cameras are warped simultaneously. To remove the artifacts due to rounding errors, a median filter is employed.

**Step 2: Removing the ghosting artifacts.**
Ghosting occurs in 3D warping when the pixels at depth discontinuities are incorrectly warped to the synthesized frame. To remove it, the algorithm

![Figure 2.5: Virtual-view rendering using warping: (a) Texture; (b) Depth map; (c) Synthesized virtual view.](image)
Figure 2.6: Virtual-view rendering using warping and blending: (a) Left texture; (b) Right texture; (c) Synthesized virtual view; (d) Left depth map; (e) Right depth map.

warps the coordinates of the discontinuities and deletes their destination coordinates.

Step 3: Blending.
The two warped images are blended together through linear weighting at every pixel position. The weight is proportional to the horizontal distance of the original cameras to the virtual-view position.

Step 4: Hole filling.
The holes in the synthesized frame correspond to residual disocclusions – parts of the scene not visible in either of the original cameras. These areas are filled using extrapolation/inpainting, which starts predominantly from the nearest background areas.

Figure 2.6 illustrates the two original camera frames, their depth maps and a virtual frame rendered between them. The details of the warping, blending and artifact minimization can be found in [65]. In terms of its quantitative perfor-
2. Efficient 3D Video Representations and Rendering Algorithms

Figure 2.7: Visualization of the rendering using disparity and occlusion maps [76]. Arrows indicate how pixels move in the parallax motion. A virtual view corresponds to a horizontal cut at a position \( v \) between the two original views. Double-headed arrows indicate that the pixel color is interpolated between two views, while the data along single-headed arrows is simply copied.

mance, this algorithm is **efficient**, in the sense of the efficiency definition from Section 2.1.2, as demonstrated in [65] by comparing its rendering quality against the state-of-the-art algorithm in [75].

### 2.2.4 Disparity map and rendering of rectified textures

The 3D data-representation presented in this section consists of textures, disparity maps and occlusion maps (Fig. 2.8). As stated in Section 1.4.1, a disparity map is defined for a pair of cameras and contains geometry information in the form of a disparity value for each pixel. In turn, the occlusion map defines the occlusion relationship between the cameras by explicitly marking the pixel positions visible in only one of the cameras.

The scene-geometry information contained in a disparity map is related to the one provided by the depth map presented in Section 2.2.3. Each pixel value \( I_{\text{disp}}(d) \) in a disparity map (in pixel) can be related to the corresponding pixel value \( I_{\text{depth}}(z) \) (in meters) in a depth map as follows [22]:

\[
I_{\text{disp}}(d) = \frac{f \cdot \Delta s}{I_{\text{depth}}(z)},
\]

where \( I_{\text{depth}}(z) \) is given in Eq. (2.4), \( f \) is the focal length and \( \Delta s \) is the camera baseline (e.g., \( \Delta s = C_x^{(1,2)} - C_x^{(1,1)} \) in Fig. 2.2(b)).

In contrast to the depth-based representation from Section 2.2.3, the presented disparity-based representation allows to generate virtual views along a chain of “weakly” calibrated cameras [76]. According to the principles of epipolar ge-
ometry [13], in the case of weak calibration, only a projective transform can be specified between adjacent cameras. This transform is defined by the fundamental matrix [13]. In [76], the epipolar geometry is estimated from the input textures directly. When the epipolar geometry is known for each camera pair, the textures are rectified to obtain horizontal epipolar lines. Next, the disparity maps are estimated for each pair of cameras in the rectified coordinate system. The rectification step makes the situation convenient for disparity estimation, since the image motion becomes 1D in the horizontal direction. The disparity map is estimated using a 1D dynamic-programming algorithm with a cost function that reduces the inconsistencies of disparity estimates between neighboring horizontal lines.

Similar to the algorithm from Section 2.2.3, the view-rendering algorithm using the disparity-based representation, enables a continuous horizontal blending between pairs of original cameras. For each virtual viewpoint, the input to the algorithm includes: (1) left texture; (2) right texture; (3) disparity map and (4) occlusion map. The algorithm creates a virtual view at an arbitrary position between the cameras. It consists of the following steps.

**Disparity-based rendering**

**Step 1:**
As a preprocessing step, the algorithm requires that the epipolar geometry is estimated for each pair of adjacent cameras and operates on two rectified textures.

**Step 2:**
Given two rectified textures corresponding to original views and a disparity map with explicit occlusion areas, the algorithm synthesizes virtual frames by distinguishing between these cases:

A. **Pixels are visible in both views.** These pixels have a corresponding disparity value $d$ assigned to it. The pixel position in the virtual view can be obtained by scaling the disparity with the view position $v$, giving a new disparity value $d' = vd$. The color values between the left and right view are also interpolated according to $v$, as indicated in Fig. 2.7 by double-headed black arrows.
B. Pixels in occlusion areas that are only visible in one view. These pixels do not have a disparity value assigned to them, as they are explicitly detected as occluded in the disparity estimation process (see marked areas in Fig. 2.7). We observe from Fig. 2.7 that the pixels in the left view are occluded by the objects positioned at their right side, while the pixels in the right view are occluded by the objects at their left side. Assuming that occluded objects are parallel to the image plane, the disparities of the background are extended into the occlusion area. Hence, for pixels in the left view, the missing disparity values are copied from the left side, while for the right view, they are copied from the right side (shaded squares in Fig. 2.7). Since these pixels are only visible in one view, we cannot interpolate the color values between the two views, but only copy them.

C. Pixels in one view whose corresponding pixels are outside the active image plane in the other view. For these pixels, a disparity value cannot be computed (see Fig. 2.7). Again assuming that objects are mostly parallel to the image plane, the disparities of the border pixels are
extended into the undefined area (see dashed lines in Fig. 2.7). Similar to the previous case, the pixel color cannot be interpolated, as the pixel is only visible in one view.

**Step 3:**

The virtual views are rendered in the rectified coordinate system, using the disparity estimates, and projected back into the original coordinate system prior to display. Carrying out the rendering on rectified textures has the advantage that only horizontal pixel shifts are involved. The rendering consists of compensating for both the translational and the rotational motion of the virtual camera. The translational motion is compensated for using the estimated disparities. The compensation of the rotational component is simulated by gradually rendering the rectification transforms.

Figure 2.8 illustrates two original camera frames, their disparity map, occlusion map and a virtual frame rendered between them. The details of the computation of the epipolar-geometry estimation, rectification transform and disparity estimation can be found in [76]. In terms of its performance, the algorithm is efficient, in the sense of the efficiency definition from Section 2.1.2. In particular, the algorithm from [76] achieves a performance comparable to the state-of-the-art algorithm that uses the same representation [77]. Unfortunately, the author of the algorithm [76] never made a quantitative performance evaluation and comparison with the state-of-the-art algorithm. Therefore, we could only make a subjective evaluation of this proposal.

### 2.2.5 Summary of virtual-view rendering

The research community has recently focused on examining the potential of explicit scene-depth representations to build efficient 3D video systems [22]. This is largely motivated by a commercial availability of multiview autostereoscopic displays that directly support the 1-texture/1-depth representation by implementing a DIBR algorithm in hardware [78]. The depth-based and the disparity-based algorithms described in Sections 2.2.3 and 2.2.4 confirm the usefulness of scene-geometry models for efficient systems. They achieve a high rendering quality and minimize the occurrence of artifacts.

---

5 “Objects” multiview sequence is kindly provided by the KDDI Corporation.
Still, it should be noted that the rendering quality depends on the quality of available disparity or depth maps. In general, an accurate estimation of scene disparity or depth is still an open research problem [59]. Currently, the best-performing algorithms have very long computation times [79]. Although hardware for depth estimation is commercially available [73], the noise level of the resulting depth maps may be insufficient for high-quality rendering [80, 81].

In comparison, light-field rendering algorithms rely on simple geometric models and are more robust to artifacts due to inaccurate model estimation. However, light field is seldom considered for 3D-system realizations in the related areas of stereoscopic 3D-TV broadcast [82] and MVC standardization [8]. We conjecture that the reason is two-fold. First, as stated in Section 1.5, a specification of 3D video data-representation and view rendering is beyond the scope of current standardization efforts. As a result, most existing research directly employs original camera streams as the 3D video data-representation and does not consider virtual-view rendering. Under these scope limitations, directly employing a light-field representation may indeed be an inefficient choice, especially without considering the possibility to compensate for light-field undersampling with an efficient rendering algorithm. Second, our literature survey in Section 2.2.2 shows that it is an open issue if the state-of-the-art algorithms can achieve a high rendering quality when the underlying light field is undersampled. We propose such an algorithm and detail on its design and evaluation in the next section.

2.3 Region-based all-in-focus light-field rendering

Our proposed algorithm follows the general approach of assigning pixels in the synthesized virtual frame to different depth layers and combining intermediate renderings at multiple focal planes to render an all-in-focus virtual frame. In

---

6At the time of finishing this thesis, some first sensors appeared giving reasonably accurate depth estimates, but only at small scene depths.
contrast to most methods surveyed in Section 2.2.2, our algorithm does not require pre-processing to determine the best depth assignment for a pixel. Instead, it makes this decision dynamically during rendering, while adapting to the 3D-scene geometry. The key to the effectiveness of the algorithm is our observation that rendering quality deteriorates if the spatial support of reconstruction filters extends over scene-depth discontinuities. Our solution is to incorporate an image-segmentation step to improve the consistent processing of objects in the same focal plane. In this section, and also in the associated figures and text, the term depth layer should be associated with the focal plane of light-field rendering. In order to relate our new algorithm in this section to existing depth-based algorithms with depth maps, we have used the term depth layer as a common ground for understanding. This holds for the remainder of this chapter. The addition of the segmentation step in the rendering algorithm ensures a high rendering quality. At the time of writing this chapter, our use of image segmentation to dynamically control the spatial support of reconstruction filters during light-field rendering is unique to the proposed algorithm.

Figure 2.9 shows the basic steps in the algorithm. For a given virtual viewpoint, multiple intermediate frames are synthesized by assuming a different depth of the focal plane for each synthesized frame, as in [69]. Each of the frames is focused on a particular part of the scene. To estimate frame areas that are in focus, we use a focus metric for each pixel in a synthesized intermediate frame. This metric is derived from the one proposed in [64]. However, in contrast to [64] where the focus metric is computed using a square template centered at each pixel, our method constrains the computation to a segmented region containing the pixel. Then, areas in focus in each intermediate frame are identified by thresholding the focus estimates. In this way, we assign pixels to different depth layers. Finally, the areas in focus are combined to synthesize the final virtual frame. In the sequel, we detail on each of the algorithm steps.

A. Depth discontinuity-dependent rendering at multiple focal planes

By positioning the focal plane at a certain depth during rendering, scene parts close to this plane are rendered in focus. In turn, due to our use of an undersampled light field, parts of the scene at depths significantly larger or smaller than the depth of the focal plane will be blurred due to aliasing and appear out-of-focus in the rendered virtual frame. Figure 2.10 illustrates rendering geometry for a
virtual view of a simple 3D scene containing only two objects, \( \text{Obj}^{(1)} \) and \( \text{Obj}^{(2)} \). For each pixel in the synthesized view \( V \), the rendering algorithm first defines the ray \( r \) corresponding to this pixel (\( r' \) in Fig. 2.10). In geometry terms, the ray \( r' \) corresponds to a line connecting the pixel and the virtual-camera center \((V_s, V_t)\) [66]. Then, the algorithm computes the intersection point of the ray \( r' \) with the focal plane. In our notation, \((u_q, v_q)\) denotes a focal plane positioned at depth \( Z_q \). Although the focal plane can be positioned at an arbitrary depth, we bound the depth of the focal plane by the minimum and maximum scene depths, \( Z_{\text{min}} \leq Z_q \leq Z_{\text{max}} \), according to the plenoptic-sampling analysis [14]. Next, the computed intersection point is used to define rays \( r_q^{(1)} \) and \( r_q^{(2)} \) that connect this intersection point and the centers of the two adjacent original cameras \((C_s^{(1)}, C_t^{(1)})\) and \((C_s^{(2)}, C_t^{(2)})\). The pixel values corresponding to rays \( r_q^{(1)} \) and \( r_q^{(2)} \) in the original cameras are used to interpolate the pixel value in the synthesized view.

As Figure 2.10 shows, if the focal plane is at depth \( Z_q = Z_{Q-2} \) that is close to the depth of \( \text{Obj}^{(1)} \) (focal plane \((u_{Q-2}, v_{Q-2})\)), the rays \( r', r_{Q-2}^{(1)} \) and \( r_{Q-2}^{(2)} \) intersect the surface of \( \text{Obj}^{(1)} \) at three closely-spaced points. In Figure 2.10, these rays are illustrated using a full line and their intersections with \( \text{Obj}^{(1)} \) are marked using a square symbol. A close spacing of the intersection points is an accurate approximation of the ideal case where all three rays \( r', r_{Q-2}^{(1)} \) and \( r_{Q-2}^{(2)} \) sample the plenoptic function at the exact same point on the object’s surface. As a result, \( \text{Obj}^{(1)} \) is rendered in focus.

Correspondingly, our algorithm positions the focal plane at depths \( Z_{\text{min}} \leq Z_1, Z_2, \ldots, Z_Q \leq Z_{\text{max}} \), such that each of the \( Q \) synthesized frames contains an
in-focus rendering of a different part of the scene. In each of the $Q$ renderings, the focal plane is positioned at a depth $Z_q$ [64], given as:

$$\frac{1}{Z_q} = \frac{1}{Z_{\text{max}}} + \frac{q - 1/2}{Q} \left( \frac{1}{Z_{\text{min}}} - \frac{1}{Z_{\text{max}}} \right). \quad (2.7)$$

Given multiple synthesized frames, the algorithm estimates the pixels in each frame that are rendered in-focus. The estimation is based on an assumption that pixels corresponding to the same scene point have similar colors. This assumption is similar to the color-constancy and color-similarity assumptions commonly made in the field of computer vision [83]. By testing for color similarity, we identify pixels and areas in a synthesized frame that are rendered in-focus. In the example in Figure 2.10, the test produces a larger similarity value in the synthesized pixels by positioning the focal plane at $Z_q = Z_{Q-2}$ compared to $Z_q = Z_{Q-1}$. The reason is that by positioning the focal plane at $Z_q = Z_{Q-1}$, the rays $r'$ and $r^{(1)}_{Q-1}$ sample the surface of $Obj^{(1)}$ at two widely spaced points. In Figure 2.10, these rays are illustrated using a dashed line and the corresponding intersection points using a circle symbol. As a result, $Obj^{(1)}$ is rendered out-of-focus, which is reflected by a low color similarity in the synthesized frame.

A challenge when relying on color similarity to estimate the in-focus pixels are scene-depth discontinuities. As Figure 2.10 shows for the case $Z_q = Z_{Q-1}$, due to the existence of a depth discontinuity from $Obj^{(1)}$ to $Obj^{(2)}$, the ray $r^{(2)}_{Q-1}$ samples a point on the surface of $Obj^{(2)}$. Since this point is occluded in the required view $(V_s, V_t)$, including this sample in the interpolation will likely result in a rendering artifact. In the sequel, we detail on how our algorithm addresses this challenge.

**B. Region-based depth-layer assignment using image segmentation**

To evaluate color similarity in a synthesized frame, our starting point is the focus metric proposed in [64]. The focus metric $f_q(i, j)$ is defined as:

$$f_q(i, j) = \sum_{-M<l,k<M} \sum \frac{\text{sub}_q(i+k, j+l)}{(2M+1)^2}. \quad (2.8)$$
Differently from [64], we calculate $\text{sub}_q(i, j)$ as:

$$
\text{sub}_q(i, j) = |A_q(i, j) - B_q(i, j)|,
$$

$$
A_q(i, j) = \min(Cr_w + Cg_w + Cb_w), \quad w \in W,
$$

$$
B_q(i, j) = \max(Cr_w + Cg_w + Cb_w), \quad w \in W.
$$

The parameters are explained as follows. The parameter $W$ denotes the width of the employed aperture filter [69]. The parameters $Cr_w, Cg_w, Cb_w$ correspond to the amplitudes of the red, green and blue components of the ray $r$ used in the interpolation of pixel $(i, j)$. Using variables $A_q(i, j)$ and $B_q(i, j)$, we keep track of the minimum and maximum of the sum of color amplitudes over different color channels. These sums are used to estimate the difference in color amplitudes among the rays used for interpolation, defined as $\text{sub}_q(i, j)$. The value of $\text{sub}_q(i, j)$ will be small if the pixel at position $(i, j)$ is in-focus and large if the pixel is out-of-focus. Although $\text{sub}_q(i, j)$ can be used as a focus metric directly, the resulting depth assignment may be noisy. Hence, $\text{sub}_q(i, j)$ is averaged over a square block of pixels of size $2M - 1$, using Eq. (2.8). By employing a square template centered at each pixel, the focus metric is computed more robustly.

The computation of the focus metric described thus far assumes that the in-focus pixels are in correspondence and that this will be reflected in a high similarity score. However, this assumption is violated at depth-discontinuity boundaries if a square template is used across a depth discontinuity, as illustrated in Figure 2.11. In this situation, pixel outliers may significantly deteriorate the overall similarity score. As a result, evaluating the focus metric for pixels close to depth-discontinuity boundary becomes difficult. This motivates us to include image segmentation in the computation of the focus metric.

We use image segmentation to obtain regions with similar colors and compute the focus metric by a template matching without crossing region boundaries. The employed segmentation algorithm is a version of normalized cuts [84]. We select this algorithm because of its robustness and the availability of an open-source implementation. In this technique, pixels in the image are represented as nodes of a weighted undirected graph $G = (V, E)$ with $V$ vertices and $E$ edges. Every pair of nodes $(i, j)$ is connected by an edge whose weight $s(i, j)$ is a function of

\footnote{We consider the assumption of per-block consistency reasonable because such an assumption is commonly made in image and video coding standards (e.g., JPEG, MPEG-4).}
Figure 2.11: Visualization of the matching process when crossing a depth-discontinuity boundary [45]: (a) Square-template matching; (b) Region-based template matching.

the similarity between nodes $i$ and $j$. The similarity is measured using colors in the image. The graph $G = (V, E)$ can be segmented into two disjoint parts $I_1$ and $I_2$, by removing the edge(s) connecting these parts. The degree of similarity between the obtained parts can be computed as the total weight of the edges that have been removed, denoted as $\text{cut}(I_1, I_2) = \sum_{u \in I_1, t \in I_2} s(u, t)$, where $u$ and $t$ are variables. The optimal partitioning of a graph is the one that minimizes this value. The algorithm uses a fraction of the total edge connections in the graph as the cost of a cut. This leads to the so-called normalized cut.

C. All-in-focus virtual-view rendering – the complete algorithm

The pseudo-code of the complete algorithm is given in Algorithm 1 and consists of the following steps.

Step 1:

The normalized-cuts segmentation algorithm is applied as follows. For efficiency, instead of applying this algorithm to segment each intermediate frame, we apply it to a frame specifically rendered for this purpose. We render this frame by positioning the focal plane at optimal depth $Z_{opt}$, as suggested by plenoptic-sampling analysis [14]. The optimal depth for a scene depends only on the minimum and maximum scene depths and is defined as:

$$\frac{1}{Z_{opt}} = \frac{1}{2} \left( \frac{1}{Z_{max}} + \frac{1}{Z_{min}} \right).$$

(2.11)
Algorithm 1: Region-based all-in-focus rendering (light-field data, $Z_{\text{min}}, Z_{\text{max}}, (i, j), W, Q$, threshold)

Generate $I_{\text{opt}}$ using $Z_{\text{opt}}$;
Segment $I_{\text{opt}}$ into regions;
foreach $q=1$ to $Q$ do
  Generate intermediate frame at focal-plane depth
  \[
  \frac{1}{Z_q} \left\langle \frac{1}{Z_{\text{max}}} + \frac{q-1/2}{Q} (\frac{1}{Z_{\text{min}}} - \frac{1}{Z_{\text{max}}}) \right\rangle;
  \]
  $\text{sub}_q \leftarrow \text{color difference for each pixel while using filter } (Z_q, (i, j), W)$;
  foreach region seg in $I_{\text{opt}}$ do
    foreach pixel p in seg do
      Calculate sum of $f_q$ over all pixels in the region;
      if sum $\leq$ threshold then
        Assign seg to $Z_q$;
        Render all pixels in region using quadrilinear filter
      else
        Perform weighted blending from all $Z_q$;
  return all-in-focus rendering

We then segment the frame rendered at $Z_{\text{opt}}$ into regions. The maximum and minimum depths are assumed to be available from the capturing process [9].

Step 2:
The algorithm generates $Q$ intermediate frames in multiple rendering passes. In each pass, one intermediate frame is generated by positioning the focal plane at depth $Z_q$ according to Eq. (2.7) and using a wide-aperture filter as in [69].

Step 3:
For each intermediate frame, we compute the metric $\text{sub}_q$ according to Eq. (2.9). Subsequently, we perform template matching within each of the regions to compute $f_q$ according to Eq. (2.8).

Step 4:
The algorithm then assigns each region to a depth layer by computing a sum of $f_q$ over all pixels in the region and comparing it to a threshold. In case the computed sum is smaller than the threshold, a depth layer is assigned and the final pixels values are calculated using a standard quadrilinear filtering
in this region [67]. Otherwise, a large value of the sum of $f_q$ indicates that a depth layer cannot be reliably assigned for this region. Correspondingly, the final pixel values in such regions are rendered by a weighted blending over all available intermediate frames, which effectively emulates the wide-aperture filtering in [69].

D. Performance evaluation

We implement the proposed rendering algorithm in Matlab and use different light-field data sets to evaluate its performance. Data set “Toys” (256 images, 320×240 pixels) is obtained from the MIT Light Field archive. Data set “Jewel” (289 images, 672×420 pixels) is captured with a 2D camera array and is accessible from the Stanford New Light Field Archive [9].

In all our experiments, we compute the focus metric in Eq. (2.8) using $M = 8$ (8x8 pixel block) and $W = 8$ (8 rays selected in a square pattern around the ray to synthesize).

We evaluate the performance of our algorithm both objectively and visually. Figure 2.12 illustrates the rendering at depth $Z_{opt}$, the final rendered frame and a comparison of the accuracy of depth layers estimated using our proposed algorithm and the algorithm in [64]. Note that the depth maps depicted in Figures 2.12 (b) and (d) are not explicitly reconstructed, but only visualized here for comparison.

For the objective comparison, we apply Root Mean Square Error (RMSE) as a metric to quantify the rendering error, which is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{I} \sum_{j=1}^{J} (V(i,j) - O(i,j))^2}{I \cdot J}}. \quad (2.12)$$

In the above, $V$ and $O$ are the synthesized virtual and original frame, respectively, and $I \cdot J$ stands for the frame resolution. This image-quality metric can be applied directly when comparing the relative performance of two rendering algorithms. However, it does not quantify the absolute quality of the rendered frames. We note that our rendering algorithm creates virtual views of the scene, i.e., frames not present in the original data set. A plausible way to verify its effectiveness is to compare the rendered virtual frame to the ground truth - the original frame.

---

8Unfortunately, the MIT data set is no longer available for download at the time of this writing.
Figure 2.12: Algorithm steps and results (“Toys” data set): (a) Rendering at $Z_{opt}$; (b) Estimated depth layers without segmentation; (c) Segmentation of the $Z_{opt}$-rendered frame; (d) Estimated depth layers with segmentation; (e) All-in-focus rendering.
from the same viewpoint. To obtain the ground-truth frames, we divide the entire data set into two subsets by assigning every even row and column from the camera array to the new input data set and every odd row and column to the ground-truth data set.

The results of RMSE comparison are shown in Table 2.1. The first column refers to our proposed algorithm and the second to our implementation of the baseline algorithm by Takahashi and Naemura [64]. For the sake of completeness, we also include the score for rendering at the optimal focal plane $Z_{opt}$ [14] in the third column in Table 2.1. We use $Q = 5$ focal-plane depths in all experiments. The results in Table 2.1 correspond to virtual frames synthesized at different positions in the array (indexed by the array row and column). A synthesized virtual view is compared to the ground-truth frame at the same position. We present only a limited number of representative scores, as we have observed a similar
2. Efficient 3D Video Representations and Rendering Algorithms

...trend for other viewpoints. Both the baseline algorithm and our proposed algorithm perform significantly better than rendering at Zopt that uses a single plane only (as could be expected). The results further show that our algorithm achieves an RMSE improvement of, on the average, 10% over the baseline algorithm. This moderate gain is partly due to the structure of the test sequence with a large uniform background. A visual comparison clearly illustrates the advantages of the proposed algorithm. Figure 2.13 shows a magnified view on the scene objects. The square-template matching in the baseline algorithm is insufficient to reproduce the complicated patterns on the object surfaces and introduces significant rendering blur. Our region-based approach produces sharper renderings in the same area, as evidenced by the ground-truth comparison.

The robustness of the color-based focus metric depends on two assumptions: (1) no occlusions occur, and (2) object surfaces are Lambertian. Nevertheless, the metric appears to be robust when used with the proposed region-based approach, as we experimentally demonstrate by rendering scenes with complex occlusion relationships (such as in “Toys”) and strong specular reflections (like in “Jewels”). We thus offer empirical evidence that our approach is not sensitive to the violation of these assumptions and has a certain degree of robustness.

We expect similar performance gains for other data sets, as the algorithm is not constrained by the accuracy of segmentation. Future work will be to adaptively optimize the selection of segmentation thresholds for a given scene, as they are currently set empirically.

2.4 Conclusions

Our work in this chapter addresses the challenge of achieving bandwidth efficiency in a 3D video streaming system by employing efficient 3D video data-representations and rendering algorithms. Specifically, the presented rendering algorithms use bandwidth-efficient, undersampled 3D-scene representations to synthesize virtual views at a quality comparable to original views and without visually-disturbing artifacts. As a result, a 3D video streaming system constructed using these algorithms will be able to provide a high rendering quality at practical bandwidth costs.

Our proposed region-based all-in-focus light-field rendering algorithm, which forms the original contribution of this chapter, renders undersampled light fields at a quality visually indistinguishable from the original views. The algorithm
achieves this by both: (1) employing rendering at multiple focal planes and (2) image segmentation to enhance the depth-layer assignment during rendering. As a result, it minimizes aliasing artifacts and maximizes the rendering quality at depth discontinuities, a previously unsolved problem in the state-of-the-art algorithms. The performance of our algorithm is evaluated with an experimental analysis of the rendering quality using publicly available data sets and a comparison with the best known all-in-focus rendering algorithm [64]. Our proposed algorithm improves rendering quality both quantitatively (RMSE reduction of 10% on average) and, more significantly, visually. We note that these results are obtained by empirically optimizing the segmentation thresholds. Future work will be to systematically optimize the selection of segmentation thresholds for a given scene.

More generally, the rendering quality achieved with the proposed algorithm using undersampled light fields suggests that light field is a viable 3D video representation for future 3D video streaming systems. Our current implementation based on wide-aperture filtering can be directly employed in a system designed for multiple-perspective autostereoscopic viewing. This will be suitable for multi-user autostereoscopic displays based on the light-field representation, e.g., [85, 3]. Alternatively, for a single-viewer system, where at most two views need to be rendered simultaneously, the algorithm can be easily adapted to use a simpler filter (e.g., a camera-skip filter as in [64]) and achieve a yet higher bandwidth efficiency.

The rendering-quality improvement of the proposed algorithm is achieved at the expense of a higher computation complexity, caused by incorporating image segmentation. In general, image segmentation can be a computationally-intensive task. In our case, segmentation is performed during rendering. Although we do not analyze the complexity of our Matlab implementation in depth, we have observed increased computation times compared to the implementation of the baseline algorithm in [64]. Still, we note that we implement a relatively simple segmentation in the form of color segmentation. Therefore, we conjecture that an optimized implementation of our algorithm will be possible without compromising real-time rendering. A recently reported hardware-accelerated implementation of the baseline algorithm achieves frame rates of up to 30 fps [86]. We believe that our algorithm, if implemented along the lines suggested in [86], can achieve similar real-time properties while providing better rendering quality.
The depth-based and disparity-based rendering algorithms presented in Sections 2.2.3 and 2.2.4, respectively, achieve a high rendering quality and confirm the usefulness of scene-geometry models for efficient virtual-view rendering. However, their integration in an efficient 3D video streaming system is still an open issue. Particularly challenging in this respect is the problem of optimizing streaming quality in a situation where the original views and scene-geometry models are not displayed directly, but used by the rendering algorithm to synthesize virtual views. We address some of these challenges in subsequent chapters of this thesis (Chapters 4, 5 and 6).

Importantly, as all the 3D video representations and algorithms presented in this chapter can be conceptually viewed as sampling and reconstruction of the plenoptic function [14], they should be regarded as complementary rather than as alternatives [12]. We therefore believe that a yet higher bandwidth efficiency can be achieved by combining them into a hybrid rendering algorithm. A general framework and a starting point for further research in this direction may be the methodology proposed in [87].

An optimized selection of the efficient 3D video representation to use in a given streaming system is nevertheless a difficult problem. Such a solution needs to holistically consider reconstructing, compressing, transmitting and rendering a particular representation. In subsequent chapters of this thesis, we examine these issues in depth. The optimized selection of a 3D video representation is further discussed in Chapter 3. As the quality of rendering will depend on the compression used for transmission, a joint optimization of virtual-view rendering and compression algorithms is the focus of Chapter 4. The quality of virtual views rendered using decompressed undersampled scene-representations is further analyzed in Chapter 5. Finally, a system architecture to efficiently support the entire range of viewing scenarios enabled by sampling a 3D scene with a camera array will be considered in Chapter 6.
Chapter 3

Layered Framework for Heterogeneous 3D Video Streaming

An important requirement for a 3D video streaming system is to simultaneously accommodate receivers that have heterogeneous resources or preferences. This chapter proposes a layered framework for 3D video streaming as a unifying and efficient solution to the problem of heterogeneity of receiving devices and viewing preferences. The framework is general in its applicability to unicast and multicast network architectures. The architecture components of our framework include: (1) efficient 3D video representation, (2) efficient decomposition of the 3D-scene description into information layers, where each layer conveys a single coded video signal or coded scene-geometry data and (3) rendering of virtual views. Heterogeneous receivers can select the number of layers to receive for view rendering, depending on the availability of resources or viewing preferences. To demonstrate the viability of the proposed architecture, we implement a 3D video streaming prototype and show that heterogeneous autostereoscopic 3D displays can be supported by the system. This prototype is acknowledged as one of first two stereoscopic streaming prototypes in the research community [49]. To the best of our knowledge, it is also the first 3D video streaming prototype to support virtual-view rendering.
3. Layered Framework for Heterogeneous 3D Video Streaming

3.1 Heterogeneity challenges in 3D video streaming

As stated in Section 1.3, our focus on video streaming as the 3D video delivery mechanism is based on an assumption that IP-based networks are best positioned to serve as a substrate for the gradual deployment of 3D-TV and FTV services, and also as their long-term operational environment. Deployment of 3D video services over the Internet opens access to a large client base, including a growing number of mobile users, which is important for the service acceptance.

Deployment of 3D video streaming over the Internet is challenging due to the need to transmit a potentially large amount of data for a single 3D scene and process this data for display. In our view, the main challenge is the cost of provisioning network bandwidth and computation resources. We assume that providing these resources will continue to be expensive in the future Internet, while the cost of coding and signal processing are decreasing steadily. As a result, availability of communication and computation resources at the endpoints will continue to be scarce. Importantly, the availability of these resources will also continue to be heterogeneous. Although available bandwidth and computational capabilities of the Internet endpoints will continue to grow, there will always be scenarios where the aggregate resource demand exceeds the resource supply. Thus, the resource heterogeneity and scarcity are challenges for a 3D video delivery system that aims to simultaneously accommodate many users with highly heterogeneous resources.

Resource heterogeneity is inherent in distributed systems of the size of the Internet. A 3D video streaming system (Fig. 1.1) deployed over the Internet faces heterogeneity challenges at several levels.

- **Access-bandwidth heterogeneity.** Recent measurements show that despite a steady growth in broadband access bandwidths, differences of several orders of magnitude exist among the Internet endpoints [38, 88].

- **Access-bandwidth cost heterogeneity.** The cost of broadband subscription shows a strong variation worldwide as well as a strong dependence on the broadband technology (e.g., wired or wireless) [88].

- **Path-loss heterogeneity.** Depending on time or location, the loss rates of Internet paths can vary by several orders of magnitude due to link loss and bursts of traffic demand [89].

- **Server cost and performance heterogeneity.** The cost of bandwidth for hosted
servers may vary widely over time and location [90], often affecting user-perceived throughput and latency [91].

- **Device heterogeneity.** Internet endpoints range from smartphones and tablets to servers, resulting in large differences in available bandwidth, computation power and specialized computation capabilities (e.g., availability of Graphics Processing Units (GPU) or Digital Signal Processors (DSP)) [92]. Notable differences also exist with respect to specialized display capabilities (e.g., autostereoscopic displays) and display resolution.

- **Viewing-preference heterogeneity.** A requirement for a 3D video system is that the view rendering should enhance the sense of immersion, which suggests that the service should adapt to the user and leads to heterogeneity of viewing preferences in addition.

Some of these challenges are not unique to 3D video systems and have been adequately addressed in prior work on conventional 2D video streaming. In our view, this holds for the heterogeneity of access bandwidths and server performance, as well as for the heterogeneity of endpoint devices in general [93, 94]. In contrast, our work focuses on the heterogeneity challenges specific to the new capabilities of 3D video systems, as compared to conventional 2D video systems. In Section 1.4, we identify two such distinct capabilities: stereoscopic rendering on specialized display devices and multiple-perspective viewing with user interaction. Correspondingly, our work directly addresses challenges pertaining to display-device heterogeneity and viewing-preference heterogeneity. In addition, we give due consideration to other heterogeneity challenges although our work does not address them directly in this chapter. Specifically, the main objectives of our work in this chapter are as follows:

- Achieve a real-time 3D video streaming-performance (including multiple-perspective viewing) on a number of commodity Internet-endpoints and a good rendering quality on different state-of-the-art stereoscopic displays.

- Accommodate the viewing-preference heterogeneity by providing a solution for users to communicate their preferences and adapting the streamed content accordingly.

This chapter is structured as follows. Section 3.2 analyzes the open issues in related work and puts our work in context. Section 3.3 presents our solution: it
introduces the layering concept and our choice of a 3D video data-representation. In Section 3.4 we present the implemented prototype and detail on the software components and their integration in the system. Section 3.5 highlights the main points of the presented work.

### 3.2 Related work

Due to scarcity of prior research on 3D video streaming, the literature on 3D video lacks examples of unifying frameworks for multi-stream transmission. Correspondingly, the heterogeneity problem in 3D video streaming has not been addressed yet. To motivate our research in this area, we resort to an analysis of application areas that are related to 3D video streaming, most notably, tele-immersion and virtual-reality systems.

**A. Tele-immersion systems.** Tele-presence services augment traditional multi-party Internet conferencing with high-quality video rendering [2]. Stereoscopic and multiview 3D video conferencing services are next-generation systems that will enhance natural representation of all conference participants [95, 96]. A transmission model where a sender transmits all available camera streams to receivers is common in immersive teleconferencing systems, which employ multicamera setups at each participant’s site [97]. The choice of the delivery model is clearly dictated by the application – to create a continuous sense of immersion, all available streams must be rendered at the same time. In this way, receivers can quickly switch between the available viewpoints. Due to high bandwidth requirements, such systems are typically implemented over high-bandwidth WAN networks (e.g., Internet2). Still, compression [98] and congestion-control algorithms [99] are important in practice. Optimized strategies have also been proposed to exploit unequal importance of the individual streams and maximize the quality of experience for the user [100]. The same delivery model has also been considered for projector-based immersion systems [3]. The camera streams are encoded using the MPEG-2 standard and rendered on a multi-projector system with lenticular screens. The work in [3] focuses on projector-system design and rendering, rather than on transmission aspects.

**B. Virtual-reality systems.** Distributed collaboration and virtual-reality services enhance productivity, or a sense of immersion, by visualizing distributed
data at very high rates [5]. A number of approaches in the field of computer graphics address the reconstruction, transmission and rendering of 3D-scene representations [8]. Most related to our work are the approaches that focus on modeling and reconstruction of real-world scenes [7]. The main focus in this line of work is 3D-scene modeling to reconstruct an accurate volumetric model for a selected scene object. The idea behind the approach is to automatically reconstruct such a model from the available camera images instead of constructing it by hand. Different from our assumptions, the requirement is that the reconstructed models can be embedded in new virtual scenes and rendered in different lighting conditions. Most commonly, a scene with a single human figure is considered. Aiming at real-time rendering, Matusik et al. [101] recover an incomplete, view-dependent visual-hull model of an object offline, distributing the processing over several PCs. Rendering results show a good quality, while the hulls are extracted with as few as four cameras. However, rendering is complex and runs in parallel over four PCs to obtain real-time properties. Würmlin et al. [102] also take an approach based on visual-hull reconstruction, but use dynamic-point samples as the rendering primitive instead of triangular meshes. Experiments show convincing results for rendering of human actors captured with eight cameras. The resulting dynamic-point representation can be encoded and transmitted in real-time for visualization [103]. Carranza et al. [104] assume that a generic human-body model in the form of a triangular mesh is available, and focus on capturing the object’s motion in the scene by tracking this model throughout the video sequence. For rendering, view- and time-dependent textures captured by the cameras are mapped onto the model; however, the motion capture is performed offline.

C. Stereoscopic and multiview video systems. An early work on stereoscopic video transmission is Smile!, an end-to-end system for display of stereo streams [105]. The system employs the JPEG still-image coding to encode the streams and uses active shutter-glasses for display. The main focus is on protocols for signalling the two streams as belonging to the same session. Contemporary to our work, [18] and [19] study stereoscopic video systems. However, these systems do not support multiple-perspective viewing and focus on optimized encoding for streaming, without detailing on the transmission. A system that supports multiple viewpoints and a number of multiview special effects, is presented by Lou et al. [106]. A similar system is discussed in [107] and [108], where a peer-to-peer overlay architecture is proposed for multiview video delivery. These systems lack
the virtual-view rendering functionality and display the received streams directly. As a result, the set of available viewpoints is limited to original cameras. Most similar to our work is the system of Kimata et al. [109]. This system consists of a large number of closely-spaced cameras so that multiple views can be created by view rendering in the ray-space. When the receiver requests a viewpoint change, the server transmits only the rays required to create the desired virtual viewpoint. However, the transmission aspects are not detailed in the paper.

3.3 Unifying framework for 3D video streaming

Our proposal addresses the device and viewing-preference heterogeneity challenges in 3D video streaming by jointly considering three problem dimensions: (1) 3D video data-representation and rendering, (2) compression algorithm and (3) 3D video streaming model.

1. As demonstrated in Chapter 2, the quality of virtual views rendered using a 3D video representation depends on the quality of its reconstruction and the number of textures that are blended together. The inherent trade-off with 3D video representations is between the computation cost of accurate reconstruction and the resulting bandwidth savings for rendering. This suggests that the most suitable 3D video data-representation is the one that optimally matches the resource level and preferences of each user. However, finding an optimal representation for a large number of heterogeneous users will limit the scalability of practical systems. This raises a problem of defining a 3D data-representation that is suited to such a system.

2. The quality of virtual views rendered from decompressed 3D video data depends on the compression efficiency of the employed coding algorithm. This presents the following trade-off. Although a compression algorithm specialized for the given 3D video representation may result in higher efficiency, a standard coding algorithm may achieve a satisfactory compression performance for a range of heterogeneous 3D video representations.

3. As a 3D video streaming service should be available to all endpoints with Internet connectivity, the streaming layer should not place restrictions on their location. This includes service deployment over the two dominant network architectures – unicast and multicast – and a variety of network
paths. A unifying video streaming model should be sufficiently flexible to support a real-time transport of the compressed 3D video representation in such heterogeneous transmission environments.

### 3.3.1 Layered streaming

Layered streaming is the first component of our proposed solution. Information layering is a well-known guideline for the design of multicast streaming systems [94]. It is commonly used to simplify the problem of multicast heterogeneity by allowing the system to select the number of video layers to transmit based on receiver capabilities. This design requires video streams to be compressed into a number of temporal, spatial or quality bitstream layers, e.g., using scalable video coding algorithms [94], where scalability refers to the number of layers.

We propose to apply the information-layering concept in 3D video streaming such that each information layer conveys a single texture stream or a scene-geometry stream. The number of layers to receive for a single virtual view can be matched to the average available bandwidth, computation power or display capabilities. Heterogeneous receivers can select the number of layers to receive for view rendering and optimize the rendering quality under their resource constraints or capabilities. In view of a large number of possible optimizations, such decisions are best implemented at run-time, for each receiver independently. A layered streaming supports this model well, thus offering a form of quality scalability.

It is important to note that unlike layered multicast, the layered 3D video streaming does not require scalable video coding, although it can benefit from it if available. In other words, quality scalability in layered 3D video streaming is achieved by using a layered 3D-scene description and not by compressing a monolithic 3D-scene description into bitstream layers.

The proposed layered streaming supports a multicast architecture if the individual texture and geometry layers are transmitted on different multicast sessions. A receiver can join a number of multicast sessions at run-time. The layered streaming is also flexible in that it permits different combinations of texture and geometry layers to share a multicast session.

### 3.3.2 Selective view streaming

The second component of our proposal is selective view streaming. It follows directly from the requirement for accommodating the heterogeneity of viewing pref-
erences. In a 3D video session with many users, some viewpoints will be requested frequently, and others not at all, entirely driven by the user interest. Transmitting a representation of the entire 3D scene to each user, as in tele-immersive systems surveyed in Section 3.2, is inefficient in that it treats all viewpoints as equally important. Instead, an on-demand transmission would be more efficient. Selective view streaming implements such on-demand transmission. A user either interactively selects a new viewpoint, or his movements are continuously tracked and the displayed content automatically adjusted. This design can be implemented with a feedback channel to communicate the user interest in real-time.

The selective view streaming supports a multicast architecture when combined with the previously proposed layered streaming. A user navigating a certain part of a 3D scene joins only the multicast sessions that carry texture and geometry layers for that part.

### 3.3.3 3D data-representation selection

In this section, we motivate our choice of the 3D video data-representation for texture and geometry layers. We first perform a comparative analysis of potentially suitable representations. We compare these representations with respect to their efficiency and suitability for the proposed layered streaming. In particular, we compare efficient 3D video data-representations presented in Chapter 2 against the representations commonly used in tele-immersive and virtual-reality systems from Section 3.2. We note that our analysis is necessarily qualitative, since a quantitative evaluation is difficult in the absence of directly comparable data and thus beyond the scope of this analysis.

Both the local 3D video representations in Section 3.2 and the global representations in Chapter 2 can be used in the proposed layered streaming. The local representations only describe the scene from a subset of viewpoints. Thus, a local, view-dependent model is a natural fit for the selective view streaming and lends itself to a layered representation. The global modeling approaches reconstruct a geometric model consistent with all input cameras. As such, global models may require simplification for view-dependent streaming, especially if the scene is complex [110]. This is why very complex global models are often converted to local models for transmission [111].

Bandwidth efficiency of different 3D video data-representations can be analyzed using a general notion that an accurate scene model reduces the amount
of data for rendering [12]. Both the local and the global representations rely on scene models, either explicitly or implicitly. Global models include a geometric model of scene surfaces and in some cases models of surface reflectance properties and light sources [7]. The possibility of reusing the reconstructed models in novel scenes under different lighting is certainly attractive. However, our work targets a different application space. Instead of focusing on novel-scene rendering, our main goal is to enable interactive exploration of a remote scene. Due to our focus on multiple-perspective viewing under original lighting, our interest in these models is limited to surface geometry. It is worth pointing out that the best-performing algorithms for object-geometry reconstruction (in terms of accuracy of geometry reconstruction) have long computation times, often amounting to tens of minutes per video frame [59].

The computational complexity of reconstructing and rendering accurate global models of real-world scenes often limits the applicability of most of these models in real-time systems. Global models may be attractive in 3D video streaming due to their potential to significantly reduce the required bandwidth. However, current global modeling prototypes mostly focus on individual scene objects in order to limit the computation complexity [8]. The ability of these techniques to scale with the scene complexity (e.g., the number of objects in the scene) is questionable. Therefore, we do not regard global models as a cost-effective solution for real-time systems in the near future. Their high reconstruction and rendering costs limit their applicability in 3D video streaming applications where we are after an enhanced immersive experience at an affordable cost.

In contrast, local reconstruction models generally have a lower complexity and the hardware for certain local geometry representations such as depth, is already available [73]. We regard the local 3D video representations as an efficient and cost-effective solution for our layered streaming framework. Their integration in the proposed framework is presented in the next section.

### 3.3.4 System architecture proposal

All local 3D video representations from Chapter 2 are suitable for use with the proposed layered streaming. We continue our exposition with a focus on disparity-based (Section 2.2.4) and depth-based representations (Section 2.2.3), since they are used in our streaming prototype. However, the general characteristics of local 3D representations equally hold for the light-field representation from Section 2.3.
Figure 3.1: Unifying layered framework for 3D video streaming.

Figure 3.1 shows a realization of the proposed unifying streaming framework. Its main components and characteristics are as follows.

- **Efficient 3D video representation.** A 3D video representation consists of a number of video streams and the local scene-geometry streams in the form of disparity or depth maps. Geometry information in a depth stream involves a depth value for each pixel in every frame of the original stream, whereas a disparity representation contains a disparity value for each pixel. The number of views and the associated disparity or depth maps can be extended dynamically to support the multiple-perspective viewing.

- **Standards-based video compression.** The state-of-the-art video coding standards, MPEG-4, H.264/MPEG-4 AVC and MVC are readily applicable to encode these 3D video data-representations for streaming. By treating each texture, depth or disparity stream as a conventional 2D video, a standard coding algorithm can be applied to each stream independently (MPEG-4, H.264/MPEG-4 AVC) or with inter-stream prediction (MVC).

- **Virtual-view rendering.** Multiple nearby views of a scene can be generated at the receiver by warping the original pixels into the new viewpoint, based on the depth or disparity map. The warping will be effective for narrow-field view rendering, as in stereoscopic rendering [82]. For wide-angle perspective
changes, two solutions to the “dissocclusion” problem are proposed. One solution is to extend the depth-based or disparity-based representation with explicit disocclusion-filling information. The other is to use the texture and geometry information from one or more nearby viewpoints and render virtual views with the algorithm presented in Section 2.2.4 or that in Section 2.2.3. The rendering may be further improved by sending additional descriptive data for very complex scenes, such as occlusion masks, edge masks or contours of key image-objects (as discussed in Chapter 2). These additional data can be considered optional enhancement layers.

- *Layered, selective view streaming.* Following the layering principle, multiple texture, geometry and enhancement streams can be transmitted dynamically and on demand. A dynamic selection of the streams at any given time can be adaptively determined by the view-rendering algorithm, taking into account the user interaction and the resources available.

### 3.4 Streaming prototype

Our analysis thus far argues that the proposed layered framework is an efficient solution to accommodate the heterogeneity of devices and viewing preferences in 3D video streaming. To support this claim, we build a streaming prototype according to the proposed framework. The prototype uses an efficient 3D video representation consisting of texture, geometry layers (depth or disparity maps) and optional scene-description layers. It employs an efficient standards-based compression and a real-time view-rendering algorithm.

Prototyping a streaming system for a new video representation is challenging. The main challenge is that several software components to build such a system are generally not available. We address this challenge by developing our own software for the required components or by relying on open-source software and extending it where necessary. The following list indicates our actions for specific components.

- No software is available for virtual-view rendering using depth or disparity maps, despite a long-time interest in the research community. We address this challenge by developing our own implementation of the rendering algorithm in Chapter 2.2.4 and adapting it for use in the streaming system.
Common software frameworks for video coding lack support for MVC and multi-stream coding in general \footnote{At the time of this writing, there is still no support for the MVC standard in common multimedia frameworks and libraries.}. To decode the texture streams, we use our own MPEG-4 decoder developed on top of software libraries available through the ffmpeg project [112]. We use another instance of this decoder for the disparity or depth streams.

Common streaming frameworks have insufficient support for multi-stream transmission and reception. For streaming, we extend the open-source live555 library [113] to support a simultaneous transmission of multiple texture and geometry streams.

Autostereoscopic displays lack standardized driver interfaces to support video playout from a PC in a unified way. As a result, different display types require different formats of the video input. We believe that due to novelty of the technology, manufacturers still favor proprietary solutions. This requires us to implement display adaptation, a software component that converts the rendered video frames to the format required by the given display type.

We assume that suitable components for the reconstruction of geometry and optional scene-description layers at the sender end (Fig. 3.1) are available from previous work. We justify this assumption as follows. For the reconstruction of disparity and occlusion maps, a number of algorithms are available through the software provided at Middlebury Stereo Vision Page [79]. For the reconstruction of depth maps, implementations of several state-of-the-art algorithms are available and surveyed in [59]. In addition, hardware for depth sensing is becoming widely available [73]. Likewise, reconstruction of additional scene-description layers such as edge masks and contours of image objects can be performed using robust edge detection and segmentation algorithms, respectively [114]. Finally, although the accuracy of reconstructing the scene-geometry layers [59, 60, 61] and the additional layers [114] are still open problems in general, we assume that by using rendering algorithms such as those described in Sections 2.2.3 and 2.2.4, we can ensure a sufficient quality of synthesized views.

In the sequel, we detail on our solutions to each of the above challenges. In particular, we describe the implemented components for video transmission, user
interaction, decoding and rendering, as well as their integration in the system. For ease of exposition, we describe a two-layer version of the multi-layer system depicted in Figure 3.1. Without a loss of generality, we will assume that one layer refers to a texture stream and the other to its accompanying depth stream. In Section 3.4.5, we describe and experimentally evaluate alternative layer instantiations.

### 3.4.1 Sender end: layer-reconstruction and compression

At the sender end, our proposed framework (Fig. 3.1) requires components for depth-stream reconstruction and compression of texture and depth streams.

As discussed in Section 3.4, the component for depth-stream reconstruction is assumed available from previous work and we do not detail on its implementation here. For completeness, we provide Table 3.1 that summarizes our test sequences and illustrates that some sequences used in our experiments already contain the required geometry information. This is the case for the “Ballet” multiview sequence that already contains the depth information, which is computed at the capturing stage. For other test sequences where this information is not available, we use existing algorithms to reconstruct the geometry layers and the additional scene-description layers. This is the case for the “Objects” multiview sequence (Table 3.1), where we employ the algorithm in [76] to reconstruct the disparity and occlusion maps.

To compress the streams, we independently encode the texture and depth streams using the ffmpeg codec [112] and store them in separate files. We configure the codec to use only I- and P-frames and employ its default rate control algorithm to obtain constant-bitrate (CBR) streams. The main ffmpeg parameter values used for encoding are given in Table 3.1. We note that optimized rate allocations for texture and depth streams are not our focus here and will be detailed in Chapters 4 and 5. We also note that for stereoscopic streaming that uses left- and right-eye streams – as is the case for our experiments with the “Aqua” sequence in Table 3.1 – a dependent coding with inter-stream prediction would likely achieve a more efficient compression than our independent coding. As shown in [21] for MVC, the compression gains of inter-stream coding increase as the camera distance decreases. The camera baseline of stereoscopic video is relatively small, in the context of multiview sequences, as tested in [21]. We therefore expect further efficiency gains from inter-stream coding and plan to
incorporate MVC into our real-time system once efficient MVC implementations become available.

### 3.4.2 Selective streaming with user interaction via a feedback channel

To enable interactive scene viewing, we implement a feedback channel for the user. We currently only implement support for viewpoint control using standard-PC input devices.

User sends a request for a viewpoint change by moving the mouse pointer inside of the video display-window. The generated events are transmitted to the sender, including the current pointer position in display-window coordinates (in our current implementation, the “x-coordinate” only).

The sender maintains a mapping between the receiver’s display-window coordinates and the coordinates of the capturing cameras. When a request arrives at the sender, we first determine if the receiver coordinates match one of the original cameras and if so, we transmit the requested stream directly. Otherwise, we send the texture and depth streams needed to render the virtual view with the requested display coordinates.

Eq. (2.5) in Section 2.2.3 represents this mapping between the display and scene coordinates and is repeated here in a shortened form for ease of exposition:

$$\lambda \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} C_{i3\times4} & \begin{bmatrix} R & \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} 13\times3 & -t \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix} \end{bmatrix}.$$

#### Table 3.1: Test sequences and encoding parameters.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Left+Right</th>
<th>Texture+Disparity</th>
<th>Texture+Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Aqua”</td>
<td>“Objects”</td>
<td>“Ballet”</td>
<td></td>
</tr>
<tr>
<td>Frame rate</td>
<td>10 fps</td>
<td>20 fps</td>
<td>25 fps</td>
</tr>
<tr>
<td>Resolution</td>
<td>720×288 pixels</td>
<td>320×240 pixels</td>
<td>1024×768 pixels</td>
</tr>
<tr>
<td>Video codec</td>
<td>mpeg4</td>
<td>mpeg4</td>
<td>mpeg4</td>
</tr>
<tr>
<td>Rate control</td>
<td>CBR</td>
<td>CBR</td>
<td>CBR</td>
</tr>
<tr>
<td>Bitrate</td>
<td>5 Mb/s</td>
<td>1 Mb/s</td>
<td>10 Mb/s</td>
</tr>
</tbody>
</table>
The rendering process maps each visible 3D-scene point \((x_s, y_s, z_s, 1)^T\) to a point \((x_i, y_i, 1)^T\) in display coordinates. For example, by mapping the pointer position in display coordinates \((x_i^{(N)}, y_i^{(N)}, 1)^T\) to 3D-scene space according to the inverse of Eq. (3.1), virtual views \(V^{(0)}, \ldots, V^{(N-1)}\) at arbitrary positions \((x_i^{V(N)}, y_i^{V(N)}, 1)^T\) can be synthesized, where \(x_i^{V(N)}\) and \(y_i^{V(N)}\) stand for the coordinates of the center of the view \(V^{(N)}\).

3.4.3 Multi-stream transmission

The transmission components of our prototype integrate a set of standardized streaming protocols. The focus of this section is to explain our protocol choices, the encapsulation of layers and to describe how we combined existing protocol libraries to implement a multi-stream transmission.

The protocol stack of our prototype is illustrated in Figure 3.2. It consists of UDP at the transport layer, Real-Time Transport Protocol (RTP) [115] as the transport-layer extension and Real-Time Streaming Protocol (RTSP) [116] at the application layer. The properties of these protocols that are important for our prototype are summarized below.

- UDP provides an unreliable datagram-transport service. UDP packets may be lost or duplicated in the network, delivered to the destination out of order, or arrive corrupted.

- RTP is a general-purpose multimedia transmission protocol that closely integrates with the application layer. We use RTP as it permits extensions for new video representations through header extensions and RTP payload-formatting. The RTP payload-format specifications are IETF-standards documents, defining how the bitstream corresponding to a particular video content-type is encapsulated in RTP \(^2\). RTP’s flexibility in supporting newly-defined formats greatly simplifies the encapsulation of the depth data and their simultaneous transmission with the texture data, as a part of the same streaming session. In addition, RTP functionalities important for our system include packet sequence-numbering and packet time stamping. Our receiver implementation relies on sequence numbers for de-packetization of the multi-stream data. In turn, a standardized representation of time stamps in

\(^2\)Definition of an RTP Payload Format for the transmission of 3D video is beyond the scope of our work. The ongoing standardization efforts are surveyed in [117] and [118].
multiple streams simplifies the inter-stream synchronization and integration with common video players.

- RTSP is an application-layer protocol that provides access and presentation control for real-time streaming sessions. RTSP is general in that it supports both live and on-demand streaming and can be deployed in unicast and multicast architectures.

Our choice of unreliable transmission protocols may seem an odd design decision in face of TCP’s dominant share of roughly 90% of Internet video streaming today [26, 36]. However, this decision serves our stated goal of supporting both today’s dominant network architectures – unicast and multicast. If deployed in a unicast architecture over the public Internet, our layered streaming would use RTP over TCP.

For the implementation of the protocol stack, we use the live555 [113] software library as a basis. We motivate this choice as follows. First, live555 provides standards-compliant implementations of RTP and RTSP. Second, its source code is open and the code base is mature and stable. Third, it has a highly optimized implementation. Fourth, as the library is already used in the transmission subsystem of several popular video players (e.g., MPlayer [119]), we can reuse and extend its well-defined interfaces for passing the compressed bitstreams to decoding and rendering components. In this way, we ensure compatibility with the existing video-player interfaces in that the texture and depth bitstreams can be decoded and virtual views rendered in any video player that implements suitable decoding and rendering algorithms.
Figure 3.3: Multi-stream transmission using live555 library; transmission of texture and depth video frames over a LAN connection is illustrated.

The live555 library is object-oriented and uses an event-based architecture. These two architectural properties are important for our system implementation. First, live555’s object-oriented design simplifies implementation of protocol extensions required to encapsulate the depth data and transmit it alongside the texture data. By providing an abstract interface and an implementation of common functionalities shared by many existing protocols, new protocols or protocol extensions can be developed in live555 by implementing specific new functionalities on top of the common code base and reusing the common interfaces. Second, at run time, real-time protocol operation is ensured by relying on events to synchronize access to objects in memory. The use of event-based synchronization in live555, instead of the more common thread-based synchronization, allows to accurately control packet transmission- and reception-times without relying on OS-schedulers. This property of live555 is important for our system due to the need to support multiple real-time streams, as opposed to most current systems that typically only need to support two (video and audio).

Figure 3.3 illustrates our implementation of the protocol stack based on the live555 library. The live555’s TaskScheduler is the central control unit in our event-based protocol architecture. It collects the events generated by various processing components in the datapath, enqueues them and dispatches them to registered event-handlers. In our implementation, the texture bitstream and the depth bitstream follow datapaths with similar processing components. This datapath consists of components for encapsulation and packetization at the sender end, and de-packetization at the receiver end. The functionality of each individual component is implemented as follows.

- **FramedSource**, or more generally, a **Source** in live555 is an abstraction of a component that indicates the start of a datapath. A FramedSource object
encapsulates the bitstream stored locally, or generated in real-time by encoding a live video source. It permits encapsulations of arbitrary content types and coding formats, with the only requirement being that the bitstream must be structured as a sequence of frames, where all data in one frame is sampled at the same time instant. Since in our 3D data-representation each texture frame has an accompanying depth frame, the texture- and depth-sampling instants can be assumed identical. In general, this means that a single FramedSource object can be used to encapsulate both the texture and the depth data. However, in our implementation, we use separate FramedSource objects for texture and depth, consistent with the layer semantics of our proposed streaming framework (Fig. 3.1). Importantly, this separation also gives us flexibility in terms of implementing different coding algorithms for the depth in the future.

The encapsulation implemented with FramedSource must be specialized for the given coding format. In our case of using the MPEG-4 encoder, an instance of FramedSource is created such that it contains the metadata specific to the MPEG-4 standard. This metadata includes a representation of sampling time stamps, frame sizes and frame types (I- or P-frames in our case). Subsequently, the texture and depth frames and their metadata encapsulated in FramedSource objects are passed to the next stage and a corresponding event is dispatched to the TaskScheduler.

- RTPSink component receives the texture and depth frames encapsulated as FramedSource objects and sends them to the network. The “Sink” semantics in live555 reflect that this component ends our datapath at the sender.

\[^3\]An encapsulation that uses a single FramedSource object would be appropriate for stereoscopic video formats that by design combine the left- and right-eye data into a single frame (e.g., side-by-side or interlaced layouts [20]).
end. An RTPSink object implements packetization of the FramedSource-encapsulated data, sets the packet-header fields, passes the packets to the OS through the Socket API (UDP) and dispatches control events to the TaskScheduler.

The packetization performed by RTPSink is fully RTP-compliant. The most important aspect of RTP packetization for our system is “application-level framing” [115], a packetization strategy where the bitstream is packetized as a sequence of independently processable units. This is achieved in RTP by encoding the presentation- and decoding-related metadata in RTP packet headers. Since we use an MPEG-4 codec for both the texture and the depth, we set packet headers as specified in MPEG-4 RTP payload format [120]. Figure 3.4 depicts the structure of RTP packets for texture and depth and Table 3.2 summarizes packet-header fields according to [115]. We set these header fields based on the metadata in texture and depth FramedSource objects.

The last step implemented by RTPSink is passing the packets to the OS for transmission. For this step, we follow the layering principle in Fig. 3.1 and transmit the texture and depth streams as separate RTP sessions, i.e., using different RTP ports [120]. We rely on existing code in live555 to pass the packets to the OS. We control packet transmission-times at two timescales as follows. At the short timescale – the order of inter-packet intervals – the times are controlled by the TaskScheduler, according to its internal event scheduling. We do not modify this scheduling, as it proved sufficiently fast in our experiments. At the long timescale – the order of inter-frame intervals – we implement a simple round-robin discipline by interspersing the packetizations of texture and depth frames before generating events to the TaskScheduler. In this way, we ensure that a texture frame and its depth frame are transmitted at approximately the same time instants and rely on the TaskScheduler to manage packet timings at the short timescale. Our experiments confirm that these timings are correctly managed in all cases.

- **RTPSource** is an abstraction of the de-packetization component at the receiver end. RTPSource processes the texture and depth packets that arrive at a listening UDP socket to reassemble the frames, and forwards the complete frames to the next component in the datapath. It relies on RTP packet-header fields to correctly de-packetize the data and recover the associated
<table>
<thead>
<tr>
<th>Header Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Payload type (PT)</strong></td>
<td>Identifies the content type and coding of the payload carried inside the packet; texture and depth are different content types in our implementation, although the coding standard used is the same for both.</td>
</tr>
<tr>
<td><strong>Sequence number</strong></td>
<td>Allows the receiver to reconstruct the original transmit order of the packets in case they arrive out of order; in our implementation, we use different sequence-number intervals for texture and depth packets.</td>
</tr>
<tr>
<td><strong>Packet time stamp</strong></td>
<td>Reflects the sampling instant of the first octet in a texture or depth packet.</td>
</tr>
<tr>
<td><strong>Synchronization source</strong></td>
<td>A random number ensuring that no two synchronization sources within the same RTP session have the same SSRC; note that in our system texture and depth are transmitted as different RTP sessions.</td>
</tr>
</tbody>
</table>

**Table 3.2: RTP packet-header fields for texture and depth.**

metadata. To achieve this, RTPSource differentiates texture packets from depth packets by the PT (payload type) field in the header of incoming packets (Fig. 3.4). Once a complete frame is assembled, an event is generated to signal the availability of data to the next component.

- FileSink object ends the datapath at the receiver end by first encapsulating the texture and depth frames and their metadata and then passing them to decoding components. The encapsulation should be performed such that it complies with the interface of a given decoder. In our current implementation, we use a named file-pipe for the communication between the transmission and the decoding components. Correspondingly, a FileSink object is the most appropriate encapsulation.

The described components at the sender and receiver ends are integrated into an end-to-end client-server system using the RTSP implementation available in live555. We use RTSP as a first step in each streaming test, to establish a connection between our client and the server. After the connection is established,
the texture and depth streams follow the datapath consisting of the above components.

3.4.4 Receiver end: decoding, view rendering and display adaptation

At the receiver end, we implement components for decoding, rendering and display adaptation (Fig. 3.1). In this section, we detail on their implementation.

We base our implementation of the decoding component on two software libraries available through the ffmpeg project [112]: libavcodec and libavformat. We choose these libraries because they support real-time MPEG-4 decoding, have an efficient implementation and a stable code base, as evidenced by their inclusion in a number of widely-used video players (e.g., vlc [121] and MPlayer [119]).

The functionalities of these two libraries – important for our prototype – are as follows. The libavcodec contains a standards-compliant implementation of the MPEG-4 standard. To correctly configure the decoder for a given bitstream, the MPEG-4 requires a number of bitstream-specific coding parameters. According to the MPEG-4 standards document [16], two options exist to signal these parameters to the decoder. With the “in-band” option, the parameters are carried in the bitstream headers. In turn, the “out-of-band” option assumes that the parameters are available through external means, e.g., that they are a part of the file format used to store the bitstream, or that they are supported by the signaling channel of the communication system used to transmit the bitstream. By design, libavcodec supports both options in a unified way. Namely, it provides a standard interface to include the required coding parameters, independently of the actual signaling format. This independence is enabled by libavformat, a companion library that contains routines to extract the coding parameters from a variety of file formats and signaling channels.

Figure 3.5 illustrates our implementation of the decoding component using the libavcodec and libavformat. The input to the decoding component are the texture and depth frames encapsulated as FileSink objects. These frames and the associated metadata are passed from the transmission component to the decoding component through a named file-pipe. As the texture and depth frames are encoded independently, we need to decode the incoming sequence of alternating texture and depth frames independently from each other. Two implementation options exist. The first is to use one instance of the MPEG-4 decoder for the
Figure 3.5: Decoding using ffmpeg libraries.

Figure 3.6: Virtual-view rendering and display adaptation.

texture and another instance for the depth. The second is to use a single instance of the MPEG-4 decoder and maintain two separate prediction loops and two sets of reference frames. We choose the second option, as it has a lower memory requirement and is well supported in libavcodec through an optional definition of multiple AVContexts. Therefore, we define two AVContexts to be able to separately decode the texture and depth streams. Each AVContext is configured with the coding parameters available in the metadata encapsulated in FileSink and those we extract from bitstream headers using libavformat routines.

A sequence of decoded texture and depth frames is passed to the rendering component through a named pipe, as illustrated in Figure 3.6. In the sequel, we describe the main functionalities of the rendering component.

The rendering component synthesizes a virtual view at the position given by
the coordinates of its center, as described in Section 3.4.2. To synthesize the view, we employ a rendering algorithm appropriate for the given 3D data-representation, as detailed in Chapter 2. For the “Objects” multiview sequence (Table 3.1), we employ the algorithm introduced in Section 2.2.4. For a view to synthesize, the input to the algorithm includes: the left-texture stream, the right-texture stream, a per-pixel disparity map and an occlusion map. The algorithm synthesizes a virtual view at an arbitrary horizontal position between the left and the right view. To achieve a real-time performance, our implementation partly relies on hardware acceleration, i.e., it uses the OpenGL API [122] to take advantage of the hardware-rendering capabilities of today’s PC graphics cards. For the ‘Ballet” multiview sequence (Table 3.1), we rely on an existing implementation of a DIBR algorithm available in the stereoscopic display used in our prototype [78]. The algorithm synthesizes virtual views through 3D image warping [74].

After rendering, synthesized virtual frames are passed to the display-adaptation component, as in Figure 3.6. The display-adaptation component converts a pair of stereoscopic images to the format required by a given autostereoscopic display.

The displays employed in our prototype are lenticular autostereoscopic [11]. These displays are based on LCD technology with an overlayed lenticular sheet that refracts the emitted light in different directions. The left and right stereoscopic views are rendered separately and projected into the user’s left and right eye using the lenticular sheet. In this way, the user experiences the depth effect without a need for special glasses or head-mounted displays. We experiment with two types of lenticular displays. The first is a single-view display, where two stereoscopic images of the scene are simultaneously displayed [123]. The required input is two stereoscopic images in the vertical-interlaced format, as in Figure 3.7(a).
The second lenticular display is a multiview display that simultaneously shows 9 pairs of stereoscopic images [78]. This is achieved by constructing the lenticular sheet with 9 different viewing zones, such that in each zone a different pair of images is rendered. Figure 3.8 illustrates such a division of the display field into zones and the projection of different image pairs to user’s eyes \(^4\). The input format of this autostereoscopic display consists of a texture image and its depth map in side-by-side layout, as in Figure 3.7(b). The display internally synthesizes 9 pairs of stereoscopic images, using a variant of the DIBR algorithm in [74].

### 3.4.5 Experimental results

In this section, we describe experiments performed to validate our proposed layered streaming framework. As stated in Section 3.3, our proposed framework provides a solution for the heterogeneity challenges by jointly considering 3D data-representation and rendering, compression algorithms and streaming model. Correspondingly, we design our experimental scenarios to validate the following two aspects of the framework:

- **Real-time performance.** We demonstrate that our choice, implementation

\(^4\)Note that Figure 3.8 is provided by the display manufacturer and reproduced here to illustrate the principle of lenticular-display construction. We include this figure as is, although it contains a minor inconsistency. Namely, Figure 3.8(a) depicts a display with 9 zones, while in Figure 3.8(b) only 7 views are numbered.
and integration of system components results in a 3D video streaming system that achieves a real-time performance end-to-end.

- **Heterogeneity solution.** We demonstrate that our framework accommodates device heterogeneity and the heterogeneity of viewing preferences;

In the sequel, we detail on the implemented experimental scenarios and results. An overview of the results is given in Table 3.3.

<table>
<thead>
<tr>
<th>Experimental scenario</th>
<th>Number of layers</th>
<th>Layer resolution</th>
<th>Layer frame rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stereoscopic</td>
<td>2</td>
<td>1024×768 pixels</td>
<td>25 fps</td>
</tr>
<tr>
<td>Multiview</td>
<td>4</td>
<td>800×600 pixels</td>
<td>20 fps</td>
</tr>
</tbody>
</table>

**Table 3.3:** Proposed layered streaming achieves a real-time performance in different experimental scenarios, while using commodity hardware. In the multiview scenario, we also achieve real-time viewpoint changes with a delay of 500 ms.

**Scenario 1: Dual-layer stereoscopic video streaming**

Figure 3.9 shows the implemented dual-layer streaming.

- For stereoscopic streaming that uses the left- and right-eye stream representation (“Aqua” sequence in Table 3.1), we employ the single-view autostereoscopic display [123]. We do not perform virtual-view rendering for this sequence. For display adaptation, we convert the left and right frames to the vertical-interlaced layout (Fig. 3.7(a)).
For stereoscopic streaming that uses the texture and depth representation, (“Ballet” sequence in Table 3.1), we employ the multiview autostereoscopic display [78]. For display adaptation, we convert the texture- and depth-frames to the side-by-side layout (Fig. 3.7(b)). The display internally synthesizes 9 pairs of virtual stereoscopic images in a narrow range and displays them simultaneously.

The streaming experiments are performed on a LAN in our department. The receiver is a 3-GHz Desktop PC with a state-of-the-art graphics card 5.

We achieve real-time end-to-end performance with both sequences. The synchronization between the two instantiated layers is maintained over all system components. The depth effect while viewing a 3D scene is compelling on both autostereoscopic displays. This experiment confirms that:

- Our prototype achieves a good video quality on standard IP networks and a real-time performance on commodity hardware.
- Autostereoscopic displays with different capabilities can be supported within the same framework.

Scenario 2: Interactive multiple-perspective viewing

In this validation scenario, our streaming prototype was integrated in a streaming demonstrator developed by the authors of [52] and [56]. In this demonstrator, individual 3D video streaming and display services are composed into distributed applications in a virtual community.

Figure 3.10 shows the implemented demonstrator and its main components. The properties of the demonstrator most relevant for 3D video streaming are summarized as follows.

- **Service-oriented architecture.** The software architecture of the prototype is service-based and consists of a number of data-plane and control-plane services. **Orchestrator** is the central service that controls and coordinates the operation of other services by performing service discovery, selection and adaptation. Data-plane services under the control of the **Orchestrator** include: 3D video streaming, Display and Mouse. Control-plane services that support the operation of **Orchestrator** are: resource manager

---

5Commercially available under Nvidia GeForce 7600GS.
Figure 3.10: Deployment of our 3D video streaming prototype within the demonstrator of context-aware streaming in virtual communities [52, 56].

- Publish-Subscribe-based communication. Communication among services is implemented using a Publish-Subscribe architecture (visualized as directed lines in Figure 3.10). In this architecture, individual services subscribe to events of interest and are notified each time such an event is generated.

- Context-aware adaptation. The demonstrator performs a context-aware adaptation of applications. Orchestrator is the service that implements this adaptation for a 3D video streaming application by coordinating a number of other services. Most importantly, Orchestrator subscribes to ResourceMgt for service-selection decisions such that a service can be selected adaptively, in response to the context, i.e., the availability of resources. To make an effective adaptation decision, ResourceMgt estimates the resource capability of a number of devices in the system by subscribing to each device’s DevMan. In addition, ResourceMgt collects the information on resource demands of each active service by querying Repository for service-description information.

In the demonstrator, we implement interactive multiple-perspective 3D streaming and viewing as follows (Fig. 3.10). We use the “Objects” multiview sequence
(Table 3.1) with one modification: we increase the resolution of the sequence to 800×600 pixels by a simple up-scaling. We synthesize virtual views using the rendering algorithm described in Section 2.2.4. In this demonstrator, the viewpoint-rendering is implemented as a sender-side service, where the sender encodes the rendered frames and transmits them to the receiver. The display device is either a standard PC monitor attached to a desktop PC or a mobile phone. As this is a monoscopic multiple-perspective viewing, a display adaptation is not necessary. Figure 3.11 shows two original textures from the “Objects” sequence and one virtual view synthesized between them.

Perspective changes are initiated with a mouse. The demonstrator distinguishes between the display devices by implementing them as two different Display services and associating each of them with a controlling the Mouse service. During rendering, the user selects a different viewpoint by moving his mouse pointer. This generates events from the Mouse service that are sent to Orchestrator and 3D video streaming services. In turn, Orchestrator checks the resource capability and demands for this event. If the resource demand exceeds the capability, an adaptation is needed and performed by 3D video streaming. The adaptation action is to decrease the resolution of the rendered video. The same action is performed when the user switches to another display. For instance, when the user redirects the video from the desktop PC to the mobile phone, 3D video streaming service automatically reduces the resolution of the rendered frames from 800×600 to 320×240 pixels.

The demonstrator achieves a real-time end-to-end performance and real-time viewpoint changes with a delay of 500 ms (mainly buffering delays in the video player).

This experiment confirms that:

- Our prototype achieves an end-to-end real-time multiple-perspective viewing on commodity hardware.

- Viewing-preference heterogeneity is supported by using a feedback channel to communicate the user interest and by rendering a virtual view from the user-selected viewpoint.

- Multiple-perspective viewing on monoscopic displays with different capabilities and resolutions can be supported within the same framework.
Figure 3.11: Multiple-perspective viewing: two original views and one virtual view synthesized between them.

Limitations of the prototype

The presented prototype is not as complete as our proposed framework in Figure 3.1. In particular, our current implementation and evaluation of the prototype have the following limitations.

- Our prototype implements multiple-perspective viewing by assuming that optional scene-description layers (occlusion maps, edge masks) are locally available for rendering. In a scenario that involves network transmission, these layers need to be coded. In the sequel, we provide suggestions on how a suitable coding algorithm can be selected for these layers.

Similar to our earlier observation on the applicability of state-of-the-art video coding standards for depth- and disparity-map coding (Section 3.3.4), MPEG-4, H.264/MPEG-4 AVC and MVC can be readily applied to encode the scene-description layers. In the case of an occlusion map, Figure 2.8(d) shows that occlusion areas have an image-like representation. A sequence of occlusion maps can therefore be treated as a conventional 2D video and coded using one of the above standards. For coding of edge masks or contours of image objects, a good starting point is the MPEG-4 standard [16]. Specifically, MPEG-4 represents object-shape information as alpha maps. An alpha map generated for a specific object divides the pixels in the region containing the object into those inside the object-shape contour (opaque pixels) and those outside of it (transparent pixels). As a result, alpha map is a greyscale image that approximates the objects contour. In the simplest case, the alpha map contains only two luminance levels – white for opaque and black for transparent pixels. MPEG-4 defines efficient shape-encoding
tools for such a representation [124].

- Our prototype achieves a real-time performance on commodity hardware when two layers are instantiated. Instantiating a larger number of layers could potentially have a significant effect on performance. Although we do not quantify this effect, we have performed informal evaluations by simultaneously instantiating up to 3 dual-layer streaming tests on the same hardware as in Section 3.4.5 (6 layers in total, as in Fig. 3.9). Real-time performance is achieved in all tests, suggesting that commodity desktop PCs have enough computation power for scenarios significantly more complex than those present in our experiments.

- In our evaluation, we do not present results on wide-angle viewpoint changes using texture and depth maps. The reason is that our current prototype implementation relies on the view-synthesis algorithm available in our autostereoscopic display [78]. This algorithm achieves a good rendering quality for virtual views close to the original view, as we could not observe distortions in the synthesized views. However, a wide-angle perspective change using this algorithm would lead to “dissociations” artifacts similar to those in Figure 2.5. To avoid these artifacts, a streaming system could implement the view-rendering algorithm described in Section 2.2.3. However, a real-time implementation of this algorithm was not available at the time of implementing our prototype.

### 3.5 Conclusions

In this chapter, we argue that future 3D video streaming systems over the Internet will be conditioned on the system’s ability to accommodate heterogeneity of devices and viewing preferences and to do this in an efficient way. Our analysis suggests that streaming frameworks proposed in the related areas of immersive tele-conferencing and virtual-reality lack this ability and are therefore not recommended as an architectural basis for 3D video streaming. Specifically, their reliance on global scene models limits their ability to scale the computation load with available resources or scene complexity. Likewise, transmission of complete model representations fails to consider the cost of provisioning the network bandwidth and differences of several orders of magnitude in the bandwidth available to receivers.
We propose an architecture for a streaming framework that is better suited for these heterogeneity and efficiency challenges. First, the layered streaming and the underlying layered 3D-scene description enable the system to scale the rendering quality with resource availability. Second, the selective view streaming with interactive feedback accommodates the heterogeneity of viewing preferences. Third, a solution combining a local 3D video representation, a standardized compression algorithm and a virtual-view rendering algorithm utilizes the available resources efficiently.

To support this architecture proposal, we have built a streaming prototype. A real-time system performance is achieved using texture and geometry layers, without a need for high-performance networks or distributed rendering systems. To achieve this, we overcome the lack of suitable software infrastructure in the research and industry communities. We develop our own streaming, decoding, rendering and display-adaptation components to use in this system. The prototype shows that heterogeneous autostereoscopic 3D displays can be supported by the system. This prototype is acknowledged as one of first two stereoscopic streaming prototypes in the research community [49] and, to the best of our knowledge, is also the first streaming prototype to support virtual-view rendering.

Let us come back on some details on the prototype implementation and our choice of software components to use as a basis for the prototype implementation. Our implementation of streaming and decoding components benefits greatly from the stability and performance of live555 and ffmpeg libraries, respectively. We therefore recommend these libraries as a good starting point for future work on real-time 3D video streaming systems. This general recommendation should however be accompanied with a few notable limitations. In case of ffmpeg, the current version the library relies on hardware acceleration for decoding, where parts of the decoding process are offloaded onto GPU. Such hardware acceleration considerably improves performance compared to the software-only ffmpeg library that we used at the time of implementing our prototype. However, future extensions of the library – e.g., to support inter-stream prediction as specified in the MVC standard – may prove more challenging due to a need to correctly adapt the implementation for a highly-parallel GPU architecture.

With respect to streaming components, we remark the following. In case of live555, an important scope limitation is this library’s exclusive focus on RTP/RTCP streaming using an RTSP server. A system that aims at directly implementing today’s dominant HTTP-based streaming using a Web server would
probably use a different library. An implementation of HTTP-based streaming may also consider implementing the display-adaptation component as a browser extension instead of as a standalone component. We therefore conclude that an exploration of different usage scenarios and the most appropriate software architectures and libraries is an interesting area for future work.

The proposed streaming framework is used as the architectural basis for subsequent work in this thesis. In particular, in later chapters of the thesis, we address the challenges arising when transmitting 3D video over networks with time-varying bandwidth and delay. A roadmap to later chapters is depicted in Figure 3.12 and can be summarized with the following motivating aspects.

- The transport layer of the prototype is generic in its support for unicast and multicast network architectures. Our current choice of an unreliable transport protocol is best suited for privately-managed and overprovisioned networks such as today’s IPTV [25] and DVB networks [117]. Shared public networks require a different transport layer and represent a more challenging transmission environment. Optimization of the streaming system for such an environment is presented in Chapter 4 of this thesis.

- The user-selective, on-demand transmission model may increase the interaction latency. In contrast to tele-immersive systems, the streams are no longer locally available, which may lead to large interaction latencies on long Internet paths. Optimization of the system for this scenario is the topic of Chapter 5.
Our proposal builds on efficient 3D video representations presented in Chapter 2. As already discussed, all local 3D video representations from Chapter 2 are suitable for use with the proposed layered streaming. The quality of virtual views rendered from a particular compressed 3D video data-representation depends on several factors. The proposed framework provides an architectural support for the consideration and optimization of factors that dominate the rendered video quality. Detailed analysis and algorithmic optimizations for this are presented in Chapters 4 and 5.
This chapter proposes an algorithm for bandwidth-efficient 3D video streaming over a best-effort Internet. The proposed algorithm achieves a continuous streaming and a high rendering quality, despite the variations of available bandwidth common to best-effort networks. The main contribution in this chapter is to demonstrate that quality-optimized 3D video streaming algorithms should: (1) adapt the streaming rate by explicitly considering rendering quality, (2) employ a 3D video adaptation that combines the contributions of the constituent components of a 3D video data-representation, the coding algorithm and the rendering algorithm. To this end, we implement the proposed algorithm using: (1) efficient 3D video representation and state-of-the-art algorithms for virtual-view rendering and compression, (2) streaming rate allocation based on an optimized joint texture-depth rate allocation and (3) virtual-view streaming adaptation that minimizes quality variations of computed allocations over time. The algorithm performance is evaluated using realistic simulations of Internet transmission conditions, including the impact of competing Internet traffic and real-world protocol implementations. The results demonstrate a significantly higher average video quality over quality-agnostic and conventional streaming strategies. The proposed algorithm is acknowledged in the community as the first algorithm for adaptive 3D video streaming that performs a joint optimization of the 3D data-representation, its rendering algorithm and the compression algorithm [53].
4. Bandwidth-Adaptive 3D Video Streaming

4.1 Challenges in efficient 3D video streaming over best-effort Internet

As stated in Section 1.5, a 3D video streaming service is characterized by simultaneous requirements for: (1) large bandwidth, (2) low latency and (3) large computation power. In Chapter 3, we address two challenges for future 3D video streaming in the Internet - the scarcity of bandwidth and computation resources and their heterogeneity. Our focus in this chapter is the varying availability of these resources in general and network bandwidth in particular. From the transmission point of view, the Internet is a communication channel shared statistically among multiple users. As a result, the bandwidth available to streaming services varies over time and location, due to competing traffic on shared links [125]. Despite the widely-adopted bandwidth overprovisioning practice (Section 1.4.3), transient drops in available bandwidth are frequent and lead to interruptions in today’s video streaming [31, 26]. Unfortunately, streaming interruptions cause a significant user dissatisfaction, often resulting in the user immediately terminating the session [26]. Although many state-of-the-art streaming systems employ very low-bitrate videos to increase the margin against such interruptions, this practice is fundamentally limited because it trades streaming continuity for the second important factor for user satisfaction – average video quality [26]. Both these factors are serious challenges for future 3D video streaming services that aim to enhance immersiveness by rendering remote scenes in high-quality and free of streaming interruptions.

In this chapter, we address the following challenges specific to 3D video streaming in the Internet. These questions relate to RQ 3 stated in Section 1.5 where we particularly emphasize maximizing the rendered quality under bandwidth variations.

• How can we determine a useful trade-off between streaming continuity and video quality for 3D video streaming?

• How can we algorithmically optimize this trade-off in face of a complex relationship between the quality of a transmitted 3D-scene description and the quality of rendered views of the scene?

• How do we implement streaming algorithms so as to efficiently utilize the available bandwidth in real-world transmission conditions, including unpre-
dictable variations of competing traffic and unoptimized transport protocols?

Correspondingly, our contributions in this chapter are as follows.

- To avoid streaming interruptions, we propose to dynamically adapt the streaming rate of a given 3D video data-representation by adjusting the coding rates of its multiple texture and geometry streams.

- To achieve an optimized streaming rate allocation, we propose an algorithm that dynamically allocates the coding rates of multiple texture and depth streams so as to maximize the quality of synthesized views. This algorithm is referred to as joint texture-depth allocation with temporal optimization and is our original contribution in this chapter.

- We implement and validate the proposed algorithm and provide a detailed description of each implementation aspect in order to demonstrate how our design achieves a bandwidth-utilization efficiency and usefulness in real-world deployments. We evaluate the algorithm’s performance using simulations with realistic cross-traffic and an actual transport-protocol implementation.

The sequel of this chapter is structured as follows. Section 4.2 provides an overview of the main challenges in video streaming over the Internet, followed by an overview of possible solutions for conventional 2D video streaming. This section also surveys related work in 3D video streaming. Section 4.3 presents the proposed bandwidth-adaptive 3D video streaming algorithm. In Section 4.4, we report on the algorithm implementation, evaluation methodology and performance. Section 4.5 summarizes the main points of the presented work.

4.2 Background and related work

In this section, we first introduce two main causes of performance problems in Internet video streaming today: the best-effort model (Section 4.2.1) and the transport-protocol implementation of rate fairness (Section 4.2.2). We then review state-of-the-art solutions to these problems: conservative streaming rate allocation (Section 4.2.3) and adaptive video streaming (Section 4.2.4). These solutions address issues in conventional 2D video streaming. In Section 4.2.5, we
provide a summary and derive a set of desirable properties for 3D video streaming algorithms. In Section 4.2.6, we survey recent work in the area of adaptive 3D video streaming.

4.2.1 Best-effort model and fair-rate bandwidth allocation

IP networks implement two models for supporting the applications: best-effort and Quality-of-Service (QoS) [93]. The basic difference between them is how the network resources – link bandwidth and router buffers – are shared. In the best-effort case, users share the network resources fairly. In quantitative terms, this means that whenever the aggregate rate of all flows exceeds a link capacity, each flow should get an equal share of that capacity, i.e., each flow should transmit at equal rate. This design principle is referred to as “rate fairness” [126]. In the QoS case, under the same congestion condition, some flows will be prioritized and assigned a larger share of the bottleneck-link capacity. The public Internet today predominantly implements the best-effort model. Due to a wide consensus on preserving “net neutrality” [127, 128], we assume that this model will remain dominant in the future Internet, for at least the upcoming years.

As a result of the fair resource sharing, available bandwidth, packet-loss rate and packet delays are variable, reflecting the time- and location-dependent traffic patterns in the network. With the inclusion of wireless last hops in the network, these parameters additionally depend on the varying conditions of the physical channel. The time dependency is generally unknown and hard to predict. An effective 3D video streaming algorithm needs to account for this unpredictability. Summarizing, we incorporate two aspects in this subsection, the best-effort model and the unpredictability of available bandwidth in Internet communication.

4.2.2 Transport-protocol implementation of rate fairness

In best-effort IP networks, the fair bandwidth-sharing mechanism is embedded in the transport protocol, commonly implemented at endpoints. If the sum of all flows in the network exceeds capacity, the transport protocol reacts and lowers the sending rate, thus ensuring fairness and network stability. This basic mechanism is known as congestion control [129] and is implemented in most today’s standardized transport protocols (TCP NewReno [130], CUBIC [131], DCCP [132], SCTP [133], etc.).
Standardized transport protocols implement the congestion control as follows. When a streaming application places its packets into the transport-protocols’ send buffer, the protocol sends the packets into the network at a rate governed by e.g., congestion-window size in case of TCP NewReno [130], or maximum rate in case of DCCP [132]. The exact transmission rate of a transport protocol is dependent on the particular protocol version and implementation. A close approximation of the transmission rate, suitable for performance analysis of common protocols, is given in an analytical model of TCP derived by Padhye et al. [134]. According to this model, the transmission rate of a TCP flow can be roughly approximated as:

\[ r_{tcp} = \min \left( \frac{W}{RTT}, \frac{1}{RTT \sqrt{2bp/3 + T_0 \cdot \min(1, 3\sqrt{3bp/8}) \cdot p(1 + 32p^2)}} \right), \quad (4.1) \]

where \( W \) is the maximum congestion window of the flow, \( RTT \) is the Round-Trip Time of the connection, \( p \) is the packet-loss rate on the path, \( T_0 \) is the Retransmission TimeOut interval (RTO) and \( b \) is the number of TCP segments per acknowledgement packet (ACK). According to the model, maximum transmission rate of the protocol at any time is limited, either by the application itself (left term in the Eq. (4.1)), or by the fair share of the bottleneck-link capacity (right term in Eq. (4.1)).

The model in Eq. (4.1) is particularly useful to understand the impact of the exact protocol implementation on the bandwidth available to a 3D video streaming application. The Eq. (4.1) shows that the TCP transmission rate is inversely proportional to the packet-loss rate and the \( RTT \) of the underlying network path. As a result, an increase in \( RTT \) or packet-loss rate due to an increase in competing traffic, or a link loss, reduces the available bandwidth. Most importantly, the transport protocol achieves a bandwidth-utilization efficiency that is inversely proportional to the available bandwidth and the \( RTT \) of the network path. As an illustrative example, if a 15- Mb/s TCP flow experiences a packet loss on a 12- ms path (\( RTT \)), it will need 6- s to regain the transmission rate before the loss, if no additional packets are lost during this period. A 30- Mb/s TCP flow on the same path will need 12- s to recover. Likewise, the recovery period for each of the two flows on a 24- ms path would take twice as long, 12- s and 24- s, respectively.

During the recovery period, TCP’s utilization of the available bandwidth on the path is inefficient. This inefficiency translates to lower service quality of a video streaming service implemented over TCP during the recovery period. The
4. Bandwidth-Adaptive 3D Video Streaming

duration of a service disruption is thus proportional to the bandwidth of the service and the $RTT$ of the path. In other words, it will be especially pronounced for high-bandwidth low-latency 3D video streaming services. It is therefore important that a 3D video streaming algorithm is designed and evaluated such that it matches the dynamics of bandwidth utilization of a particular transport protocol.

4.2.3 Conservative streaming rate and continuous buffering

Despite these inefficiencies, TCP is today the dominant protocol for video transport, supporting roughly 90% of Internet video streaming [26, 36]. It is widely believed that streaming services judiciously sacrifice efficiency for the ease of wide-scale service deployment over TCP [36]. In today’s Internet, such a trade-off is possible due to overprovisioning discussed in Section 1.4.3, and the combined use of a conservative streaming rate and buffering, which are discussed next.

Playout buffering is a common solution to improve the streaming continuity under unpredictable network conditions [135]. This method is particularly effective for short-term available bandwidth drops. Most today’s streaming systems delay the playback for a few seconds after the receiver’s request arrives in order to fill this buffer [31, 32, 33]. If the streaming rate stays below the decoding rate for a long enough time, i.e., equal to the buffer size in seconds, the buffer will eventually become empty and the playout will be interrupted. Thus, the larger the playout buffer, the larger the margin against bandwidth drops and the fewer playout interruptions for the user. Continuous playout buffering during a session is a straightforward extension [136]. A requirement is that the streaming rate is selected conservatively, such that it remains smaller than the average available bandwidth during the session. Finding an optimized trade-off between the conservative streaming rate and the buffer size is a difficult trade-off. In particular, the actual video coding rate and the available bandwidth are generally unknown. Today’s streaming systems adopt a pragmatic solution and opportunistically use periods of high available bandwidth to fill the buffer [31]. A recent research on video streaming over congestion-controlled connections in general, and TCP in particular, suggests that for an uninterrupted video playback, the average bandwidth available to a TCP streaming session should be at least two times larger than the streaming rate [42]. More precisely, a streaming session with a startup buffering delay of several seconds and an average streaming rate roughly two times below the average available bandwidth, should experience a negligible number of
playout interruptions over a range of paths in today’s Internet [42].

Although the use of a conservative streaming rate and several seconds of playout delay may be acceptable for low-quality Internet video today [36, 33, 32, 42], a similar trade-off may be overly expensive for future high-quality 3D video. A single 3D video streaming session requires to transmit a potentially large number of texture and geometry streams with minimal buffering. Therefore, our proposal explores a solution different from the state-of-the-art in conventional 2D video streaming. Specifically, we aim at increasing the utilization of the available bandwidth while maintaining a low playout delay. We achieve this by exploiting adaptation of a multi-stream 3D video representation.

4.2.4 Existing work on adaptive 2D video streaming

Our work is related to research proposals for adaptive streaming of conventional 2D video. Streaming adaptation is an approach known in the streaming literature [135], but rarely used in streaming algorithms for conventional 2D video [137]. With adaptive streaming, the system monitors the available bandwidth and reacts to changes by adapting the streaming rate [138]. Traditionally, adaptation aims at reducing the frequency of playout interruptions that occur in conservative strategies (“non-adaptive streaming”). The adaptation of the streaming rate can be performed by the receiver, or by the sender.

In the multimedia networking community, video adaptation is often combined with continuous buffering in Video-on-Demand (VoD) systems [136, 139, 140]. These proposals focus on accurate long-term buffer modeling and optimize the trade-off between the streaming rate and buffer fullness [42]. Adaptive streaming is also considered in the design of congestion-control protocols optimized for multimedia transmission [141, 138, 142, 143]. In the signal processing community, adaptive streaming over congestion-controlled channels is seldom discussed [137]. A notable exception is the rate adaptation in [43] based on control theory, which does not consider streaming quality.

Video adaptation currently receives interest in the context of large-scale deployments of HTTP-based streaming. In this architecture, a video stream is segmented into a sequence of small files (commonly referred to as “chunks”), which can be independently distributed over the existing Web servers and proxies, using the HTTP [39]. However, efficient adaptive streaming algorithms are still at their infancy and currently do not consider streaming quality [36, 33, 32]. An early
standardization activity in this space is DASH [40, 41]. However, a definition of adaptation algorithms is out of the scope of this standardization. Two recent research proposals that optimize the quality of HTTP chunks using the Scalable Video Coding standard (SVC) [144] are [145] and [146].

A question mostly neglected in the research literature on 2D video streaming [137] and in recent deployments of adaptive video streaming [26, 36] is the optimization of streaming quality. Namely, the efficiency of bandwidth utilization achieved by adaptive streaming is expected to translate into a higher average streaming rate and thus a higher video quality. However, the exact relationship between the streaming rate and streaming quality has received little attention in the literature.

4.2.5 Summary

As a result of fair resource sharing, best-effort model and physical channel properties, the available bandwidth is variable in today’s Internet. The bandwidth variations are generally unknown and hard to predict. Combined results of these variations and the transport-protocol inefficiency are low-quality video streaming and frequent playout interruptions. Unfortunately, they both cause a significant user dissatisfaction.

These challenges will remain in the future Internet, as evidenced by the trend of deploying large-scale video streaming systems without modifying the existing Internet architecture and the dominant transport protocols. Although this trend is understandable in view of the attractive opportunity to reuse the existing delivery infrastructures, the main challenges remain. It is therefore necessary to address these challenges at the algorithmic level, by designing efficient streaming algorithms.

However, existing streaming algorithms are only partial solutions to these challenges, or unoptimized solutions at best. Our survey of the related work shows that the state-of-the-art solutions either fail to consider all relevant factors or fail to jointly optimize across these factors. This makes it difficult to find an existing 2D video streaming algorithm with desirable properties and extend it to design an effective 3D video streaming solution.

In our view, the desirable properties of an effective 3D video streaming algorithm can be stated as follows: (1) maximizing the quality of the virtual views rendered during a session, (2) minimizing the number of playout interruptions,
(3) minimizing the buffering delay after which the playout starts. A streaming algorithm that achieves these properties will be able to maximize the experienced video quality of streamed virtual views under a limited and time-varying available bandwidth.

4.2.6 Novel aspects in adaptive 3D video streaming

In the case of 3D video adaptation, the problem of optimizing streaming quality is yet more difficult than in state-of-the-art 2D video streaming. Namely, the original views (textures) and geometry streams are not displayed directly, but used by the rendering algorithm to synthesize virtual views. As a result, two problems need to be solved. First, similar to conventional 2D video streaming, the exact relationship between streaming rate and streaming quality is unknown. Second, the relationship of streaming rates of texture and geometry streams to the quality of the synthesized views is more complex than in the case of 2D video streaming where the original views are displayed directly.

Due to the novelty of 3D video systems, literature on multiview 3D video streaming is scarce. To the best of our knowledge, and as acknowledged in the recent survey of our research area [53], our proposed algorithm is the first to consider 3D video streaming with virtual-view rendering and a joint adaptation of the texture and depth streams.

Most related to our work are adaptive streaming proposals in [100] and [18]. A multi-stream adaptation framework for tele-immersive systems is discussed in [100]. This work suggests that unequal importance of the individual streams should be exploited to maximize the user-perceived quality under a given bandwidth constraint. However, the study uses inefficient coding and rendering algorithms and does not propose a specific streaming adaptation. The adaptive stereoscopic-streaming proposal in [18] investigates bitrate savings achievable when encoding the left- and right-eye streams in different qualities or spatio-temporal resolutions. However, extensions to multiview 3D video with virtual-view rendering are not discussed in the paper.

4.3 Proposed algorithm

In this section, we propose an algorithm for bandwidth-adaptive 3D video streaming. We first provide a conceptual description of the proposed solution. We then
present our original algorithmic contribution in detail.

4.3.1 Conceptual description

A. Adaptation for efficient fair-share utilization

We propose to address the challenge of variability of available bandwidth in shared networks with adaptive 3D video streaming, as shown in Figure 4.1. Our algorithm adapts to time-varying bandwidth in order to maximize the utilization of its fair share of the network capacity. In contrast to conservative streaming rate strategies from Section 4.2.3, adaptive streaming can achieve a higher utilization of the available bandwidth. During periods of low available bandwidth, the streaming rate will be correspondingly low. When the available bandwidth recovers, the system will react by increasing the rate. In view of a high bandwidth cost of 3D video streaming, such efficient use of bandwidth will be essential for the feasibility of future system deployments. The implementation details of how the proposed algorithm modifies the streaming rate to efficiently utilize its fair share of the bandwidth are given in Section 4.4. In our proposal, adaptation is performed by adjusting the streaming rate of the 3D video data-representation, consisting of multiple texture and geometry streams needed to render the requested virtual view (Chapter 3) \(^1\). The individual streams are transmitted adaptively over a single congestion-controlled transport connection. This assumption is based on an argument that applications using multiple transport-protocol connections in parallel are unfair to other applications sharing the same network link [126]. The details of how the proposed algorithm interacts with a particular transport protocol are also described in Section 4.4.

B. Spatial and temporal quality optimization for 3D video streaming

Due to our focus on quality of virtual views rendered from decompressed 3D video data, we perform the adaptation by modifying the compression ratio of the underlying 3D data-representation. We note that the literature on streaming adaptation for conventional 2D video suggests several additional approaches that are readily applicable to 3D video adaptation. These include adapting the coding rate by modifying the frame rate, spatial resolution of the video, or a combination

\(^1\)A virtual viewpoint that coincides with one of the original cameras is a special case where only one stream is transmitted.
thereof. However, performing adaptations with virtual-view rendering is a distinct new problem in 3D video adaptation and we focus on this problem first. It is important to note that the generality of our solution is not limited by its focus on adaptivity for the case of modifying the compression ratio. Namely, a recent study on perceptual 2D video quality of combined adaptations of the compression ratio and frame rate shows that the correlation between these adaptation dimensions is low enough such that they can be optimized independently [147]. We assume that this independence remains true in the case of 3D video. Namely, since a sequence of rendered virtual frames is a 2D video sequence, the adaptations of the quality of synthesized frames and the frame rate can be optimized independently.

We optimize the quality in adaptive 3D video streaming as follows. When virtual views are rendered from decompressed 3D video data, their quality depends on the following factors: (1) coding efficiency of the employed coding standard, (2) coding rate and (3) selection of the operational Rate-Distortion (R-D) point. Optimization of adaptive 3D video streaming quality therefore requires a joint consideration of these factors in the algorithm. We refer to this optimization problem as spatial virtual-view adaptation. The details of the problem and a solution are given in Section 4.3.2. Since the variation of available bandwidth, and thus the coding rate, is unpredictable, the quality of rendered views may vary significantly over time. However, perceptual quality studies of conventional 2D video show that user satisfaction decreases if the quality variations are large and occur frequently during a session [26, 148, 149, 147]. For this reason, we propose an additional optimization procedure to avoid such quality variations. This is referred to as joint texture-depth allocation with temporal optimization.
and is our original algorithmic contribution in this chapter. The details of the complete algorithm can be found in Section 4.3.3. In Figure 4.2, we provide a roadmap to the remainder of this section in order to help the reader quickly locate this contribution.

4.3.2 Spatial virtual-view quality optimization

A. Quality metric

To express the spatial 3D video quality, we start from a metric commonly used in R-D-theoretic optimizations of image and video compression algorithms. In an R-D formulation, the distortion of a video frame is measured using the Mean Square Error (MSE). This metric objectively quantifies the distortion – quality reduction – in a compressed video frame and is defined as:

$$D = \sum_{i=1}^{I} \sum_{j=1}^{J} (C(i,j) - O(i,j))^2 / I \cdot J.$$  

(4.2)

In the above, $C$ and $O$ are the decompressed frame and the raw original frame, respectively, and $I \cdot J$ stands for the frame resolution. In video compression, the distortion $D$ is introduced in a video frame by applying a particular quantization step $q$ during encoding. Larger values of $q$ result in higher distortion and smaller coding rate $R$. This allows to select a value for $q$ that satisfies a given rate constraint $\hat{R}$. The MSE captures the distortion $D$ resulting from the quantization error and allows to compare performance of different algorithms. In practice, video codecs expose the quantization step $q$ through a quantization scale, or simply a quantizer $Q$ (H.264/MPEG-4 AVC, MPEG-4) [112]. In this way, an operational R-D point can be selected for a particular codec implementation. The dual of the MSE - Peak Signal-to-Noise Ratio (PSNR), correlates well with perceptual spatial quality and is widely used in compression literature [137]. This PSNR metric is defined as:

$$PSNR = 10 \log_{10} \left( \frac{255^2}{D} \right).$$  

(4.3)
The coding distortion of a compressed 3D video representation is related to the distortion in its texture and geometry streams. In our case of using local 3D video representations and standard video coding algorithms (Chapter 3), an operational R-D point can be selected for each stream independently. A shortcoming in this assumption is that the term “distortion” is not well defined for geometry streams and a better understanding and R-D modeling is still under development. Recent studies on the compression of depth maps show that it is better to jointly optimize the compression of the texture and the corresponding depth in terms of R-D performance [72, 150].

We measure the spatial 3D video quality as the MSE (PSNR) of a virtual view rendered from a decompressed 3D video data-representation. For ease of exposition, we assume a local 3D video data-representation consisting of \(N\) textures and \(N\) associated depth maps (Section 2.2.3). Encoding a texture stream and its associated depth stream using a quantizer \(Q\) leads to distortions \(D_t(Q)\) and \(D_d(Q)\). The distortion \(D(Q)\) of a virtual-view frame \(V_c\) rendered using the decompressed texture and depth frame (e.g., using the warping algorithm from Section 2.2.3), is defined as:

\[
D(Q) = \frac{\sum_{i=1}^{I} \sum_{j=1}^{J} (V_c(i,j) - V_o(i,j))^2}{I \cdot J}.
\]

In this equation, \(V_o\) is the virtual frame rendered at the same position as \(V_c\), but using the raw original texture and depth frames. In this way, the quality of a rendered virtual view can be directly related to the distortion of the texture and depth streams used to render it. Importantly, the quality can be controlled by a selection of quantizer \(Q\). In this way, an operational R-D point can be selected according to the target virtual-view quality and the resulting coding rate.

B. Optimization-problem formulation for spatial quality

The joint texture-depth allocation problem can now be posed as the problem of selecting a quantizer value for each of the texture and depth streams, such that the quality of a rendered virtual view is maximized. More formally, the joint texture-depth allocation can be expressed as a constrained optimization problem in the following way. Given a range of quantizers in the encoder \([Q_{min}, Q_{max}]\), find a quantizer for each of the \(N\) texture and \(N\) depth streams, such that the distortion
of the rendered virtual view is minimized under the total rate constraint:

\[
(\hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_{2N}) = \arg \min_{Q_1, Q_2, \ldots, Q_{2N}} D(Q_1, \cdots, Q_{2N}), \quad \text{s.t.} \quad \sum_{i=1}^{2N} R_i \leq \hat{R}, \quad (4.5)
\]

where \(\hat{R}\) is the maximum allowed rate of the compressed texture and depth streams and \(Q_{\min} \leq \hat{Q}_1, \hat{Q}_2, \cdots, \hat{Q}_{2N} \leq Q_{\max}\).

The optimization problem in Eq. (4.5) is a general formulation of the joint texture-depth allocation that includes arbitrary scene-viewing scenarios and rendering algorithms. It captures the effects of a quality drop in one of the streams on the final presentation quality, and the possibility to compensate for those effects by allowing a higher quality in other streams. This optimization problem is multi-dimensional and may be hard to solve in general form. As presented in Chapter 2, the quality of rendered views depends on several factors including: the rendering algorithm, the number and the selection of texture and depth streams, the quality of depth reconstruction, the employed coding standard and the scene complexity. Availability of an analytical 3D video quality-model may allow for efficient solutions. However, to the best of our knowledge, there are no analytical models of 3D video quality that account for all these factors. For this reason, experiments with the actual multiview 3D video sequences, rendering and coding algorithms are necessary.

C. Solution algorithm

Our solution to the optimization problem in Eq. (4.5) is done as follows. First, we assume that the viewing scenario is limited to continuous horizontal navigation. We use the algorithm from Section 2.2.3 to render virtual views from two texture and their corresponding depth streams. As in Chapter 3, we assume a layered streaming model and independent layer compression using a standard coding algorithm. The resulting 3D video representation consists of four streams. We further assume that there are two independent quantizers – \(Q_t\) and \(Q_d\) for the texture and depth, respectively. The same quantization factor \(Q_t\) is applied to the left and right textures, and the same \(Q_d\) to both depth streams. The general problem in Eq. (4.5) then reduces to a 2D selection of the operational R-D point \((\hat{R}_t, \hat{R}_d)\). The solution to this problem consists of two steps.
Step 1: Generating the R-D function of virtual-view rendering.

An optimized selection of a joint operational R-D point \((\hat{R}_t, \hat{R}_d)\) requires that a distortion is associated with each point in the \((R_t, R_d)\) plane. We refer to this 2D function as R-D function of virtual-view rendering.

We construct this function by iteratively encoding the texture and depth streams at multiple rates, decoding them, rendering the virtual view at each rate and computing the resulting distortion. Each iteration corresponds to one R-D point. The pseudocode is given in Algorithm 2 and the corresponding rendering configuration in Figure 4.3.

Figure 4.3 illustrates the constructed R-D function for a virtual view of the
Algorithm 2: Generate R-D function of virtual-view rendering

Data: Two textures and two depths

Render virtual view $V_o$ from raw textures and depths;

foreach Texture quantizer $Q_t = Q_{min}$ to $Q_{max}$ do
  Compress two textures at $Q_t$;
  foreach Depth quantizer $Q_d = Q_{min}$ to $Q_{max}$ do
    Compress two depths at $Q_d$;
    Render virtual view $V_c$ from decompressed textures and depths;
    Compute distortion $D$ in $V_c$ using $V_o$ as in Eq. (4.4); Assign $D$ to $RDFunction[R_t,R_d]$;
return $RDFunction[R_t,R_d]$

“Ballet” multiview sequence [62]. In this case, we use the rendering algorithm from Section 2.2.3 and an MPEG-4 Simple Profile (SP) encoder [16]. Virtual view $V_o^{(i/2)}$ positioned at the center of an arc connecting cameras $Cam^{(i)}$ and $Cam^{(i+1)}$ is used to compute the joint operating point $(\hat{R}_t, \hat{R}_d)$ for the textures $Cam^{(i)}$ and $Cam^{(i+1)}$ and their depth streams $Depth^{(i)}$ and $Depth^{(i+1)}$.

Step 2: Joint texture-depth allocation.

Formulated as a constrained optimization problem, the joint texture-depth allocation becomes a problem of finding the two “optimal” quantizers $\hat{Q}_t$ and $\hat{Q}_d$, in the set of quantizers available in our encoder $[Q_{min}, Q_{max}]$, where the combination of $\hat{Q}_t$ and $\hat{Q}_d$ satisfies:

$$ (\hat{Q}_t, \hat{Q}_d) = \arg \min_{Q_t,Q_d} D(Q_t,Q_d), \quad (4.6) $$

$$ \hat{R}_t(\hat{Q}_t) + \hat{R}_d(\hat{Q}_d) \leq \hat{R}. \quad (4.7) $$

In the above, $D$ is the distortion after the coding and rendering stages, $\hat{R}$ is the total coding rate budget, $Q_{min} \leq \hat{Q}_t, \hat{Q}_d \leq Q_{max}$, and $\hat{R}_t(\hat{Q}_t)$ and $\hat{R}_d(\hat{Q}_d)$ are the combined rates of the two texture and depth streams, respectively.

In our streaming case, the total coding rate budget $\hat{R}$ is influenced by the available bandwidth in the network. Thus, the R-D operational point obtained by solving the optimization problem in Eq. (4.5) varies over time. Since the variation of available bandwidth is unpredictable, the quality of
rendered views may vary significantly. For this reason, we propose an additional optimization procedure, which is described next.

4.3.3 Spatial and temporal virtual-view quality optimization

A. Optimization-problem formulation for spatial and temporal quality

A joint texture-depth allocation obtained by directly employing the algorithm from Section 4.3.2 results in an optimized rendering quality for a single virtual view in time. However, perceptual quality studies of conventional 2D video show that user satisfaction decreases if the quality changes are large and occur frequently during a session [26, 148, 149, 147]. Optimizing each virtual view independently is susceptible to such quality variations. We therefore propose an additional optimization that maximizes the average quality of rendered virtual views, while minimizing their quality variations over time.

Our proposal is based on an R-D-theoretic formulation for minimizing the distortion variation in a set of coding units [151]. Given a set of quantizers in the encoder $[Q_{\min}, Q_{\max}]$, the problem is to find a quantizer for each of the $K$ coding units $\hat{Q} = (\hat{Q}_2, \cdots, \hat{Q}_K)$, such that the resulting distortion variation is minimized:

$$\hat{Q} = \arg \min_{Q_2, \ldots, Q_K} \sum_{j=2}^{K} |D_j(Q_j) - D_{j-1}(\hat{Q}_{j-1})|, \text{ s.t. } \hat{R}(j, Q) \leq \hat{R}(j)_{\max},$$

(4.8)

where $\hat{R}(j)_{\max}$ is the maximum coding rate budget for the coding unit $j$, $R(j, Q)$ is the coding rate of the unit $j$ depending on the quantizer $Q$ and $Q_{\min} \leq \hat{Q}_2, \cdots, \hat{Q}_K \leq Q_{\max}$. The solution of this problem depends on the choice of $Q_1$, the quantizer selected for the first coding unit in the set $K$. We will discuss the initialization of $Q_1$ later in this section.

Approximation solutions to the general optimization problem in Eq. (4.8) are subject of research in R-D-optimized video coding. The problem is hard to solve for an arbitrary set of coding units, due to coding dependencies introduced by predictive coding algorithms at several levels (e.g., macroblock, frame, sequence) [152]. However, our problem structure allows for an efficient solution. First, each virtual view can be considered an independent coding unit in time. This follows directly from our requirement to use this optimization in a streaming system, since a dependence of coding units would lead to ineffective adapta-
tions [135] ². Second, our proposal focuses on a joint streaming rate allocation for individual layers in a given 3D video representation. A solution that requires a macroblock-level allocation would limit the scope of our adaptation to scenarios where re-encoding is possible. We therefore do not consider optimizations at this level of granularity.

B. Complete algorithm for joint texture-depth allocation and temporal optimization

Our solution to the problem in Eq. (4.8) consists of the following steps.

**Step 1: Generating the R-D function of virtual-view rendering.**
For each of the $K$ virtual views, we generate the R-D function using Algorithm 2.

**Step 2: Computing the slope of the R-D function.**
The slope of the R-D function corresponds to a distortion reduction achieved by the joint texture-depth allocation (Eq. (4.6)). For each of the $K$ views, we compute the slope of the R-D function in each of its $N - 1$ $(R_t, R_d)$ points as $-(D_{N+1} - D_N)/(R_{N+1} - R_N)$. Next, we sort all R-D points according to their slope. This is an intermediate step that allows for efficient implementation of the streaming adaptation.

**Step 3: Minimizing the distortion variation while maximizing average quality.**
We use an efficient steepest-descent algorithm, inspired by [145]. We first find a data point with the maximum slope across the $K$ views. We store a reference to this point as the current operational R-D point for the view $k$, subtract the rate at this point from the corresponding rate budget $\hat{R}(k)_{max}$ and continue the search. The iteration continues until $\hat{R}(k)_{max}$ reaches zero in one of the $K$ views, or until we reach the maximum available quality in all $K$ views. We have experimentally verified that this efficient algorithm finds a close-to-optimal solution in our tested scenarios. The optimal solution that we have explored is based on a full search in all $K$ views.

²In a multiview 3D coding system, the cameras are capturing the same scene from various angles and positions. In such a setting, the captured images are correlated so that a form of statistical dependence exists. To solve the optimization in this setting in the framework of Eq. (4.8) is beyond the scope of this chapter.
**Algorithm 3: Joint Texture-Depth Allocation with Temporal Optimization**

**Data:** RDFunction\([R_t, R_d]\) for \(K\) virtual views computed by Algorithm 2

```
foreach Virtual view \(k = 1\) to \(K\) do
  foreach \(R-D\) point \(n = 1\) to \(N - 1\) do
    \(s = \frac{D_{N+1} - D_N}{R_{N+1} - R_N}\);
    \(Slope[k, R_t, R_d] = s\);
    Sort \(Slope[k, R_t, R_d]\);
  while Maximum quality not reached in all \(K\) views do
    \(maxSlope = \arg \max_k Slope[k, R_t, R_d]\);
    \(\dot{R}(k)_{max} \leftarrow \dot{R}(k)_{max} - (R_t(k) + R_d(k))\);
    if \(\dot{R}(k)_{max} \leq 0\) then
      return Allocation\([k]\);
    else
      Allocation\([k]\) = maxSlope;
  return Allocation\([k]\)
```

The coding rate budgets \(\dot{R}(k)_{max}\) required by Algorithm 3 must be estimated online, during streaming. An effective estimation needs to account for the following factors: (1) playout buffer size, (2) available-bandwidth profile and (3) dynamics of bandwidth utilization of a particular transport protocol. Our implementation that accounts for these factors is described next.

### 4.4 Implementation and performance evaluation

In this section, we implement the proposed algorithm for adaptive 3D video streaming and describe how the implementation achieves the desirable properties stated in Section 4.2.5. We then detail on performance evaluation methodology and results. The difficulty of selecting the appropriate evaluation methodology stems from the requirement to account for a number of factors that affect algorithm performance in realistic scenarios (Section 4.3.1). This precludes applying common methodologies from the literature, since they do not consider time-varying traffic conditions and transport-protocol dynamics [137]. We therefore develop a methodology to realistically simulate Internet transmission conditions, including the impact of competing Internet traffic and real-world protocol implementations, as described in Sections 4.2.1 and 4.2.2, respectively.
4. Bandwidth-Adaptive 3D Video Streaming

4.4.1 Algorithm implementation

This section focuses on describing the methods and components newly developed to implement the joint texture-depth allocation with temporal optimization proposed in Section 4.3.3. The rest of the implementation relies on algorithms and methods developed earlier in this thesis – the layered scene representation, compression and streaming model from Chapter 3 and the virtual-view rendering algorithm from Section 2.2.3.

A. Multi-version encoding of the 3D video representation

The proposed adaptation algorithm requires that the compression ratio of texture and depth streams can be modified during streaming. We therefore need a general encoding solution that is applicable in cases where online encoder control is not possible, such as streaming of stored 3D video. Our solution is based on the method of Multiple BitRate (MBR) encodings or “versions” [137]. A video stream is compressed at several coding rates and each version is stored at the server. The appropriate version can then be dynamically selected during streaming, which is referred to as “version switching”.

We create multiple versions of texture and depth streams by encoding each stream multiple times and using a different quantization factor $Q$ for each version. During encoding, we compute the R-D function of virtual views using Algorithm 2. As in Chapter 3, we use standard video coding algorithms for compression. The texture and depth streams are compressed using an MPEG-4 SP encoder [16, 112]. Each version is generated using a different quantizer $Q$. We only use I- and P- frames and apply the same $Q$ for each frame. In this way, the frames of individual streams have a near-constant quality. As the MPEG-4 encoder offers 30 different quantizer options ($Q_2 - Q_{31}$) and we have to apply them to each stream, performing a joint rate allocation at the frame-level would have high computational and storage costs. We impose the following simplifications. First, we omit higher quantizer options $Q_{16} - Q_{31}$, as we visually determine that the quality of views rendered at these compression levels is unsatisfactory for practical use. Second, instead of generating an R-D function for each virtual frame, we use a combined R-D function for a set of frames. The combined R-D functions represent averages over sets. These two simplifications reduce the number of versions to store and the number of R-D points to evaluate in the optimization procedure.
B. Adaptation-interval selection

Our use of multiple texture and depth versions allows to separate the offline encoding from the online streaming rate allocation. The adaptation can then make a selection of the encoded versions for transmission. This is implemented in a component referred to as adaptive streaming control (Fig. 4.1), which tracks transmission conditions and performs the 3D-version switching. The adaptation objective is to adjust the streaming rate such that buffer starvation and playout interruptions are avoided. This suggests that the timescale of adaptive control should be matched to the size of the receiver buffer. In particular, the algorithm must adapt the rate quickly enough to avoid buffer starvation.

We select the size of the adaptation interval based on the stated desirable property for low 3D video playout-delay (Section 4.2.5). In Section 4.2.3, we have argued that immersiveness is best supported with a minimal buffering delay. We anticipate that for a convincing immersive experience, this delay should be on the order of hundreds of milliseconds (e.g., 200 ms –2 s) \(^3\). Our adaptation is performed at the same timescale.

To support version switching at this short timescale, we need to configure the encoding of versions accordingly. In particular, the spacing of switching points in the compressed bitstream must match the adaptation timescale. We place the switching points using the Group-of-Pictures (GOP) structure available in our codec implementation. Assuming a 25 fps video, we use GOPs of 10 frames, giving a GOP duration of 400 ms. Thus, the average and the worst-case GOP-switching latencies are 200 ms and 400 ms, respectively [153]. This ensures that the GOP size is not a bottleneck for switching, in view of the stated latency requirements. We use the same GOP structure in all versions and ensure that the switching points are aligned across streams. In this way, our adaptation takes place at GOP boundaries and the GOP is the basic adaptation unit. Correspondingly, the combined R-D function is computed as an average over the frames in a GOP.

C. Buffer-fullness control

The focus of our rate adaptation is preventing starvation of the receiver buffer. Namely, experiences with today’s large-scale streaming deployments suggest that rebuffering causes a significant user dissatisfaction [26]. Our buffer control ac-

\(^3\)In Chapter 5, we specifically propose a streaming algorithm that minimizes the interaction latency.
counts for the following factors: (1) available-bandwidth profile, (2) dynamics of the transport protocol and (3) variable bitrate (VBR) profile of the texture and depth streams, which is a consequence of encoding frames in a GOP with a constant quantizer $Q$.

To formalize the control problem, we use a leaky-bucket model for the variability of 3D video streaming rate. A leaky bucket is fully specified with its three parameters $(L, b, r)$, where $L$ is the bucket fullness, $b$ is the bucket size and $r$ is the draining rate. Assuming that in each time slot $k$, $R(k)$ bits are added to the bucket and $r(k)$ bits are read and removed from the bucket, the condition to prevent bucket overflow $L(k) > b$ can be expressed as:

$$L(k) = \max(0, L(k-1) + R(k) - r(k)).$$  \hspace{1cm} (4.9)

In our formulation, the leaky bucket is used to model the streaming buffer at the sender. By solving the control problem for bucket fullness, we solve the equivalent problem of receiver-buffer fullness. This approach is in accordance with the buffer-control model as described in the basic MPEG Systems standard $^4$. We set the spacing of time slots $k$ equal to our adaptation interval. In each adaptation interval $k$, the bucket is filled at the combined coding rates of texture and depth streams $R(k) = R_t(k) + R_d(k)$, and drains at the transmission rate $r(k)$. The transport protocol limits the maximum transmission rate to the fair-share rate $r_{fair}(k)$, thus $r(k) \leq r_{fair}(k)$. Correspondingly, in each time slot $k$, our buffer control adapts the rates $R_t(k)$ and $R_d(k)$ to ensure that the total rate $R(k)$ satisfies the condition $L(k) \leq b$, where $L(k)$ is given in Eq. (4.9). This control ensures that the sender buffer does not overflow and equivalently, that the receiver buffer does not underflow.

We emphasize that our leaky-bucket formulation differs from those commonly found in the literature. In our formulation, the control is performed over multiple video streams (texture and depth) and operates at a GOP-timescale. Further, the bucket drains at a fair-share rate that is stochastic. This is in strong contrast with a constant rate assumed for codec rate control [151] and VBR-control in ATM networks [154].

$^4$This is effectively a sender-based adaptation. Although a receiver-based adaptation is possible, it would require that R-D functions of virtual views are known to the receiver.
D. Joint texture-depth allocation and temporal quality control

Given multiple 3D video versions and their combined R-D functions, the proposed quality-optimized adaptive 3D-streaming algorithm is implemented as follows.

**Step 1: Estimating the transmission rate \( r(k) \).**

The transmission rate \( r(k) \) is a required parameter for the buffer control in Eq. (4.9). As the transport protocol estimates the available bandwidth in each \( RTT \), we typically obtain several bandwidth samples in each adaptation interval \( [k, k + 1] \). Our estimate of \( r(k) \) is computed from these samples and used in the next adaptation interval. Thus, we assume that at our short adaptation timescales, \( r(k - 1) \) will be a good predictor for \( r(k) \). We base this decision on findings in [156], which suggest that the available bandwidth has good predictability on timescales of up to 1 s. Specifically, we maintain a moving average \( r(k)_{avg} \) over a number of samples received within an adaptation interval [157]. The number of samples can be computed as a ratio \( \frac{400 \text{ ms}}{RTT_{(ms)}} \), assuming the adaptation interval is 400 ms. If a received rate sample is smaller than \( r(k)_{avg} \), we use the sample as \( r(k) \) directly. Otherwise, we use \( r(k)_{avg} \) as an estimate of \( r(k) \). This estimation is explained as follows. A small sampled value suggests a decreasing trend in the available bandwidth. By substituting this value in Eq. (4.9) directly, our adaptation is fast and avoids a potential buffer underflow during the next adaptation interval. Otherwise, by substituting \( r(k)_{avg} \) in Eq. (4.9), we gradually follow the trend in available bandwidth.

**Step 2: Computing the coding rate budgets \( \dot{R}(k)_{max} \).**

The Algorithm 3 requires to estimate the coding rate budgets \( \dot{R}(k)_{max} \) in each of the \( K \) views, i.e., future adaptation intervals \( k \). We compute this budget as a rate \( \dot{R}(k) \) that satisfies the condition \( L(k) \leq b \) in each of the \( K \) intervals, where \( L(k) \) is given in Eq. (4.9). For this computation, we assume bandwidth estimates \( r(k - 1) = r(k) = \cdots = r(k + K - 1) \). With this, we assume that the transmission rate from the network, will be equal in all \( K \) intervals.

---

5Modern operating systems contain extensions to expose network-state information to applications [155]. Alternatively, this information can be estimated using standardized [115] or proprietary application-layer protocols.
Step 3: Joint texture-depth allocation and temporal optimization.
We compute the joint texture-depth allocations in each of the $K$ intervals using Algorithm 3. The above assumption on the constancy of bandwidth estimates may be violated in practice, due to unpredictability of bandwidth variations when $K$ is large [156]. However, this is not a limitation for our algorithm, since it is executed in every adaptation interval $k$. In case of an abrupt drop of available bandwidth, the algorithm quickly reduces the rate to prevent buffer underflow, according to Eq. (4.9). In the next adaptation interval, the algorithm computes a new allocation to reduce the quality variation over the next $K$ intervals, based on the most recent available-bandwidth estimates. As the number of stored R-D points is low and the optimization is performed at GOP-timescale, the complexity will not be significant in practice.

4.4.2 Performance evaluation
Our evaluation methodology is designed such that performance of the proposed algorithm can be assessed in realistic transmission scenarios. As discussed in Section 4.1, transmission conditions in the best-effort Internet are governed by the impact of competing traffic on shared links and congestion-control implementation in the actual transport protocol. It is important to note that these factors impact Internet transmissions at packet level. Specifically, abrupt increases in competing traffic may lead to increased packet-queueing delays and potentially to packet loss due to overflow of router buffers. Likewise, the fair-rate transmission of the transport protocol controls the number of packets sent into the network at any given time. Eq. (4.1) captures the impact of these factors in an analytical model and illustrates the resulting packet-level dependence. However, as the work in [42] shows, the estimation of parameters in Eq. (4.1) and the resulting impact on a streaming transmission requires a complex analysis in realistic scenarios. For this reason, a common evaluation methodology in 2D video streaming research is to generalize the details of the available bandwidth profile and transport-protocol dynamics. With this simplification, the transmission conditions observed by a streaming application can be modeled as a channel of constant bandwidth, constant $RTT$ and with a statistically-derived distribution of packet-loss rate [137]. We regard this methodology as inadequate for the evaluation of our proposed algorithm. Instead, we design a simulation methodology that allows our algorithm
to observe realistic competing traffic and an actual protocol implementation. This includes experiments with actual 3D video sequences, rendering and coding algorithms. Our methodology combines a packet-level network simulation (\textit{ns-2} [158]) with a trace-driven simulation of video transmissions (Evalvid-RA tool-set [157]).

### A. Packet-level simulation with \textit{ns-2}

Although it is difficult to accurately reproduce the entire range of transmission conditions in the Internet [159], the \textit{ns-2} simulation allows to evaluate the algorithm performance under a range of realistic scenarios in a controlled setting. The \textit{ns-2} simulator supports our specific evaluation goals as follows.

- The \textit{ns-2} allows to simulate end-to-end path dynamics, using the concept of cross-traffic generators that capture the essential properties of traffic measured on real Internet links. By varying the cross-traffic levels and the distribution of flows on a network path, we can simulate a range of transmission conditions. This allows to create time-varying available bandwidth conditions for which our algorithm is designed. Even a simple network topology (e.g., “dumbbell” [160]) with cross-traffic is sufficient to realistically simulate conditions on an end-to-end network path [160].

- The realism of simulation scenarios is further supported by the ability to configure access- and core-link capacities and base \textit{RTT}s of different links in the simulator.

- The \textit{ns-2} contains tested and validated implementations of common Internet transport protocols (TCP, TFRC, etc.) and router queue-management algorithms (FIFO, RED [160], etc.). This allows to simulate the combined impact of time-varying queueing delays (\textit{RTT}), packet-loss rates (\(p\)) and the actual congestion-control algorithm, given analytically in Eq. (4.1). In this way, we can obtain accurate bandwidth-utilization profiles of transport protocols.

- Recent research on Internet congestion control has defined a range of simulation configurations recommended for evaluation of new protocol proposals [161]. These configurations focus on creating scenarios in which the dynamics of a particular protocol can be best analyzed. Since exactly this factor has a major influence on the bandwidth available to a 3D video streaming
application (Section 4.1), these scenarios are directly relevant for our algorithm evaluation. The settings for our ns-2 network simulation are given in the sequel.

B. Trace-driven video-transmission simulation

To simulate an adaptive 3D video transmission with ns-2, we extend the Evalvid-RA framework [157]. Evalvid-RA complements the ns-2 with a possibility to simulate rate-adaptive video transmissions. This allows to realistically evaluate interactions between the streaming rate and the actual transport protocol, an aspect that is often neglected in state-of-the-art streaming systems [36, 33]. Instead of transmitting the actual encoded bitstream in ns-2, Evalvid-RA uses trace files. A trace file of a compressed video sequence is a text-file that contains a bitstream description – most importantly, a list of frames with their sizes in bytes, frame type (I-, P-, B-) and the quantizer setting $Q$. Based on this description, Evalvid-RA creates the actual ns-2 packets. We extend the Evalvid-RA to support a synchronized transmission of multiple trace files, each corresponding to our compressed texture and depth streams. With this extension, we have the ability to send data from 4 streams in a single adaptation interval, corresponding to 2 texture and 2 depth streams. Further, we implement the leaky-bucket control over these streams, as in Section 4.4.1.

When a simulation starts, the trace files are read frame-by-frame and the created packets are placed in transport protocol’s send buffer (TFRC in our simulations). During the simulation, we use the extended Evalvid-RA to implement the proposed algorithm, as in Section 4.4.1. After the simulation, we perform the following post-processing steps: (1) reassembly of video frames from the received packets for all 4 streams, (2) decoding, and (3) rendering of virtual views.

C. Performance metric

As our algorithm is designed to optimize the quality of rendered virtual views, we use a video-quality metric derived from Eq. (4.4). We have used the MSE as

---

6 We do not consider distortion due to packet loss, since congestion control can be effected without actually dropping packets, e.g., by employing the Explicit Congestion Notification (ECN) through packet marking [162]. The ECN option is enabled in our network simulations with TFRC. This does not limit the realism of our simulation, since the impact of packet marking on the transmission rate is the same as if the packet was lost (as in Eq. (4.1)).
a video-quality metric for the performance evaluation. The MSE quantifies the quality reduction in an end-to-end streaming system and is defined as:

\[
MSE = \frac{\sum_{i=1}^{I} \sum_{j=1}^{J} (V_r(i, j) - V_o(i, j))^2}{I \cdot J}.
\]

In the above, \(V_r\) is the virtual frame rendered from the received decompressed texture and depth frames and \(V_o\) is defined as in Eq. (4.4). This spatial quality metric can be applied directly when comparing the relative performance of two streaming algorithms. By computing the average MSE (or its dual, PSNR) for all frames in a received video stream, the relative performance of streaming strategies can be compared. Besides using MSE, we have verified subjectively that the reconstruction quality was not eroded by severe local degradations.

**D. Simulation setup**

In our experiments, we simulate a repeated transmission of a single virtual view from the “Ballet” multiview sequence under time-varying transmission conditions. This sequence is recorded with 8 cameras, containing 100 frames in a resolution of 1024×768 pixels [62]. We create the virtual view at the center of an arc connecting \(Cam^{(0)}\) and \(Cam^{(1)}\) and refer to it as \(V^{(0/2)}\). We transmit this 100-frame sequence 8 times in succession, thus giving the total simulation time of 32 s. The cross-traffic is simulated using the \(ns-2\)’s cross-traffic generators as well as Evalvid-RA’s own ability to generate multiple video streams injected in the network from different endpoints. We instantiate a total of eight 3D video streams, one of which is the focus of our performance study, while the other seven streams are considered cross-traffic. Therefore, the bandwidth conditions on our simulated network path are highly time-varying, since it is effectively a path with low statistical multiplexing simultaneously traversed by several VBR streams. This scenario illustrates fair-bandwidth sharing of an access link, which is considered the most challenging for today’s streaming algorithms [36]. The access link capacities of all 3D video sources are set as 40 Mb/s, to ensure that these links are not transmission bottlenecks. The streams are then transmitted over a 64 Mb/s - bottleneck link simultaneously traversed by one long-lived TCP flow and another Web flow, both of which represent ”background traffic” [161]. The bottleneck router implements an RED queue-management algorithm with queue size configured as the bandwidth-delay product of the bottleneck link [161]. We use a simple “dumbbell” topology with two routers. The TFRC congestion-control algorithm
is used in all experiments and the ECN option is enabled. Table 4.1 summarizes the main parameter values used in our simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual view</td>
<td>$V^{(0.1)}$ “Ballet” sequence</td>
</tr>
<tr>
<td>Frame rate</td>
<td>25 fps</td>
</tr>
<tr>
<td>Frame resolution</td>
<td>1024×768 pixels</td>
</tr>
<tr>
<td>GOP size</td>
<td>10</td>
</tr>
<tr>
<td>Adaptation interval $[k, k + 1]$</td>
<td>400 ms</td>
</tr>
<tr>
<td>Quality-optimization interval $[K, K + 1]$</td>
<td>1.2 s</td>
</tr>
<tr>
<td>Congestion-control algorithm</td>
<td>TFRC</td>
</tr>
<tr>
<td>Queue management</td>
<td>RED with ECN</td>
</tr>
<tr>
<td>Simulation duration</td>
<td>32 s</td>
</tr>
</tbody>
</table>

**Table 4.1:** Parameters for the performance evaluation.

### E. Results

At the start of the simulation, one 3D video stream is active and shares the link capacity with the TCP flow and the Web flow, each started at a random time during the first half of the experiment. Our results focus on the performance in the stable state [161]. Each simulation is repeated 20 times and average performance results are presented. This removes the statistical bias inherent in the ns-2.

At time $t=16$ s (Frame 400), we introduce cross-traffic consisting of seven 3D video streams that stay in the simulation until $t=24$ s (Frame 600). Therefore, the intervals (0 s - 16 s) and (24 s - 32 s) roughly illustrate the steady-state operation of the algorithm. The interval (16 s - 24 s) shows its behavior during bursty traffic arrivals [161].

Figure 4.4 demonstrates the temporal-adaptation capability of the proposed algorithm. The TFRC connection first goes through a “slow start” phase – it starts transmitting packets at a low rate and gradually increases the rate until settling at its fair share. Our streaming adaptation follows this trend by selecting low-quality encodings and then gradually increases the quality as TFRC increases the transmission rate. The streaming rate and the average 3D video quality during this period are lower than in the steady state. As the available bandwidth increases in large steps from one adaptation interval to the next, the temporal quality optimization adjusts accordingly. The steady state is reached after ap-
Figure 4.4: Adaptive 3D video streaming under dynamic bandwidth conditions. The quantization level is overlayed to illustrate the quality-adaptation principle; it represents the quantization level averaged over 4 streams.

proximately 4 s (Frame 100) from the beginning of the simulation. During the steady-state period, the available bandwidth is stable and the average quality is near the maximum value. At $t=16$ s (Frame 400), we start a cross-traffic burst. The quality drop is visible only after a few seconds, as the cross-traffic streams themselves are going through a TFRC slow-start phase. This period is characterized by large bandwidth variations from one adaptation interval to the next, because the streams are simultaneously probing for bandwidth. Overall, the available bandwidth varies significantly. Our algorithm adapts by gradually selecting lower-quality encodings and keeps decreasing the quality until settling at the new fair share of the bottleneck link. At $t=24$ s (Frame 600), we stop the cross-traffic streams. The algorithm gradually increases the quality until settling at the streaming rate and quality prior to the burst.

In Figure 4.5 (a), we compare the quality achieved with the proposed algorithm against a state-of-the-art conservative-rate allocation. We repeat the scenario of inserting 7 competing video streams and compare against a conservative strategy that transmits at an average rate equal to 50% of the average available bandwidth
4. Bandwidth-Adaptive 3D Video Streaming

![Graph showing PSNR comparison]

Figure 4.5: (a) Proposed adaptive 3D video streaming outperforms streaming with a conservative rate allocation. The gain is by 2.11 dB on average; (b) Proposed joint texture-depth allocation improves the quality by 0.73 dB compared to quality-agnostic rate allocation of texture and depth streams.
during the session. This percentage is according to the recommendations in [42]. The average gain of adaptive streaming, involved with the 20 experiments, measured over 800 frames is 2.11 dB. This gain is significant enough to support our claims that: (1) adaptive 3D video streaming leads to efficient bandwidth utilization and (2) the increase in bandwidth utilization translates to a higher quality. We note that this gain is quantified using an average PSNR over the entire sequence. As such, it does not represent the perceptual quality gain achieved with our temporal quality optimization. Our informal viewing tests confirm that the quality is varying smoothly over time, without disturbing quality changes.

Figure 4.5 (b) shows the benefits of the proposed joint texture-depth allocation. We compare our optimized allocation against an allocation where equal streaming rate is allocated to each of the texture and depth streams. The results show that the optimized allocation provides a performance improvement of 0.73 dB on average, measured over 800 frames. This illustrates the benefits of our joint optimization over quality-agnostic rate allocations. At this point, we would like to comment on the significance of the achieved gain with respect to the subjective video quality. One may argue that the gain of 0.73 dB is imperceptible at high rendering qualities of 39-40 dB. We note that our proposed algorithm achieves a gain of 0.73 dB on the average, during the entire session. At lower streaming rates (corresponding to, e.g., 36-37 dB), a difference of 1 dB is significant in terms of the visibility of artifacts. Namely, at these low rates, mosquito-noise type of artifacts along the object contours are visible and are visually disturbing. Therefore, a careful optimization of texture and depth rates in this region can bring significant perceptual advantages. However, we currently lack good perceptual metrics to evaluate the impact of such low-quality periods on the user satisfaction during the entire session. This explanation indicates the value of the proposed algorithm under bandwidth variations. At the time of performing the work in this chapter, there were only few multiview 3D sequences of sufficient quality to perform the experiments. This explains the limited set of experiments in this chapter.

Let us comment on the experiment in Figure 4.5. It is typical that an adaptive rate control algorithm outperforms a non-adaptive algorithm. At the time of conducting these experiments, rate control was still under study for 2D video streaming, as it is even today. It is only recently that the research community moved from R-D model-based optimization for constant bandwidth channels to time-varying congestion-controlled channels. Since our signals are 3D and contain depth, we have favored the experiment with rate control for 3D video signals,
where the control is based on the network with time-varying behavior.

4.5 Conclusions

In this chapter, we address the quality challenge posed to 3D video streaming by the Internet bandwidth-variability. The variations of available bandwidth lead to frequent streaming interruptions that cause a significant user dissatisfaction. However, state-of-the-art solutions either lack a consideration of all factors relevant for video streaming quality, or lack a joint optimization across these factors. The problem is even more difficult in multiview 3D video streaming, where an optimized trade-off between streaming continuity and video quality needs to consider the relationship between the quality of a transmitted 3D-scene description and the quality of rendered views of the scene.

We address this challenge by proposing a bandwidth-adaptive 3D video streaming algorithm. To avoid streaming interruptions, we propose to dynamically adapt the streaming rate of a given 3D video data-representation by adjusting the coding rates of its multiple texture and geometry streams. In addition, to achieve an optimized streaming rate allocation, our algorithm dynamically allocates the coding rates of multiple texture and depth streams so as to maximize the quality of synthesized views. Our analysis, implementation and simulations show that the proposed algorithm is useful for quality-optimized streaming of 3D video on dynamic best-effort paths. The proposed algorithm achieves this due to its following properties.

- It utilizes its fair-bandwidth share efficiently by adapting the 3D video streaming rate to time-varying available bandwidth.

- The algorithm translates this bandwidth efficiency into a higher streaming quality by employing an optimized coding rate allocation. The proposed allocation maximizes the adaptive 3D video streaming quality in the sense of the spatial and temporal quality of rendered virtual views. The spatial quality is optimized with a joint selection of operational R-D points for texture and depth streams related to the same virtual view. The temporal optimization adapts the selected R-D points to minimize the distortion variation and thus quality changes over time. These two optimizations are combined in our proposed algorithm, referred to as joint texture-depth allocation with temporal optimization. The final rate allocations are obtained
as solutions to R-D-theoretic optimization problems, by jointly considering the compression and the rendering algorithm.

We evaluate the algorithm performance in highly realistic simulation scenarios of adaptive 3D video transmission. Our methodology consists of employing state-of-the-art algorithms for compression and rendering, as well as a packet-based simulation of dynamic network conditions with cross-traffic and real transport protocols. The experimental results validate the following aspects of our proposal.

- The proposed algorithm avoids rebuffering events by using short adaptation intervals that allow the streaming rate adaptation to quickly match the combined dynamics of available bandwidth and congestion-control implementation in transport protocols.

- In the absence of other 3D video streaming algorithms in the community, we compare our algorithm against a baseline algorithm implemented according to today’s best practices in conventional 2D video streaming. The results show that the proposed bandwidth adaptation achieves significant quality gains over state-of-the-art streaming rate allocations. In our experiment, we demonstrate that gains of up to 2 dB in average video quality are possible.

- Our results also demonstrate the gains of the proposed joint texture-depth rate allocation over an allocation that considers the texture and depth streams as equally important for the rendering quality. We show that average gains of up to 0.7 dB are possible with our algorithm. This finding is inline with coding experiments without streaming [72].

Although the exact numerical gains are scenario-dependent, we conjecture that our demonstrated gains represent a lower bound of the achievable gains in practice. Namely, the “Ballet” sequence used in our experiments is particularly challenging for virtual-view rendering due to its large camera baseline. As a result, the adaptation range achievable by modifying the quantization factor is relatively low and limited to about 2 dB. We believe that further experiments with simpler sequences would show yet larger gains of joint texture-depth allocations. As a related note, in our experiments, the texture and depth streams are encoded independently from each other. Although a coding with inter-stream prediction (MVC) demonstrates compression gains for multiview 3D video sequences in general, the work in [21] shows that these gains decrease as the camera baseline increases. In the context of
results presented in [21], the “Ballet” sequence has a very large camera baseline. In our experience with large-baseline sequences, the primary factor contributing to spatial quality of rendered virtual views is the rendering algorithm, rather than the compression algorithm. We therefore expect that the demonstrated gains of the proposed adaptive streaming algorithm remain valid – both qualitatively and quantitatively – for inter-stream coding using MVC.

The proposed algorithm is acknowledged in the community as the first algorithm for adaptive 3D video streaming that performs a joint optimization of the 3D data-representation, its rendering algorithm and the compression algorithm [53]. Therefore, our contribution is the first step towards the goal of achieving an efficient and high-quality multiview 3D video streaming in the future Internet. We hope that the performance gains demonstrated in our work will motivate further research on efficient streaming algorithms. To achieve the efficiency goal, further work is needed on accurate modeling of R-D functions of virtual views for general multiview scenes, which are scene-specific in our case. We also believe that further progress in this area will require to depart from simplistic assumptions on the dynamics of real Internet transmission conditions. As our work shows, holistic solutions that jointly consider the transport, encoding and rendering will be needed to make efficient 3D video streaming a reality.

The work in this chapter is based on efficient 3D video representations presented in Chapter 2. Our joint consideration of the rendering and compression components in general, and the R-D functions of virtual views in particular, suggests that an optimization of the end-to-end system should be scene-dependent. This also implies that the choice of a multiview 3D video representation to use in a given system may need to consider the exact capturing and viewing conditions. In this chapter we also rely on the layered, selective streaming model from Chapter 3. In turn, our work in this chapter extends the streaming prototype presented in Chapter 3 with an efficient adaptation capability. Further, in Chapter 5, we show the benefits of 3D video adaptations for a low-latency 3D video streaming.
Chapter 5

Interactive Low-Latency 3D Video Streaming

In this chapter, we propose a 3D video streaming algorithm that achieves a low interaction latency and high rendering quality, even in streaming systems with large and time-varying system delay. Our main contribution is to demonstrate that future 3D video streaming systems should employ streaming algorithms that: (1) explicitly minimize user-perceived latency, (2) adapt to navigation patterns of the user and the available bandwidth and (3) explicitly control rendering quality during 3D-scene navigation. The proposed algorithm achieves these properties as follows. First, it prefetches texture and depth streams in order to reduce the latency perceived, while a user is switching between multiple views of a 3D scene. To optimize the prefetching rate, we analytically derive a user-navigation model and use this model to estimate the required streams and minimize unnecessary prefetching. Second, the algorithm adapts the 3D video streaming rate – including the prefetching rate – to increase its bandwidth efficiency on Internet paths with time-varying delay and bandwidth. Third, the algorithm minimizes quality variations among multiple, consecutively rendered views of the 3D scene. We implement the proposed algorithm in a network simulator and demonstrate that it achieves a given target delay while maximizing the rendering quality. We also provide a visual evaluation of the rendering results to show that we achieve a sufficiently high rendering quality. To the best of our knowledge, we are the first to propose a complete interactive multiview 3D video streaming algorithm that achieves these properties.
5. Interactive Low-Latency 3D Video Streaming

5.1 User interaction and scene navigation in 3D video streaming

As stated in Section 1.5, a 3D video streaming service is characterized by simultaneous requirements for: (1) large bandwidth and (2) low latency. The focus of Chapter 4 is on optimizing the bandwidth-utilization efficiency and quality of 3D video streaming in face of time-varying available bandwidth in the best-effort Internet. In this chapter, we extend the results from Chapter 4 to include a solution for the other service requirement – low user-perceived latency. This chapter focuses on optimizing the user-perceived latency in face of large and time-varying end-to-end delays in streaming systems deployed over the best-effort Internet.

The importance of providing a low-latency interactivity in 3D video streaming is motivated as follows. As discussed in Section 1.5, interaction with rendered 3D scenes is important for creating a sense of immersion and regarded as the main advantage of future 3D video systems compared to state-of-the-art 2D video systems. The major way in which interactivity enhances the sense of immersion is allowing users to navigate 3D scenes. In multiview 3D video systems, this is enabled by multiple-perspective viewing that allows users to move their viewpoint around objects of interest, explore a remote environment from multiple viewpoints, or track events of interest in a 3D scene. To support such a navigation, a 3D video system allows a user to select a new viewpoint explicitly, or tracks his movements and continuously adjusts the rendered presentation. In such scenarios, it is important to render the requested views with a low latency, so as to avoid unnatural delays between a user request and displaying the requested view. Therefore, a low-latency interactivity is required to increase the realism of presentation and its immersiveness.

However, a low-latency interaction with a remote service is challenging in the best-effort Internet. This is documented in measurement studies of today’s interactive systems, most notably Internet services that provide Web and Video-on-Demand (VoD) content. First, these studies show that large response latencies are common in several large-scale systems [163, 164, 165]. Second, due to a dependence on system load, the response latency also varies over time [166]. The large and time-varying response latencies adversely affect user experience to the extent that they are viewed by Internet content-providers as a main challenge for user retention [91]. Unfortunately, analyses of the trends in system design for Internet services suggest the latencies are likely to remain a major challenge in
the future [167].

In this chapter, we address the following challenges related to large response latencies in interactive multiview 3D video streaming:

- How can we provide users with a low interaction latency in face of large and time-varying end-to-end delays in 3D video streaming systems deployed over the Internet?

- How can we design and implement solutions that achieve a low interaction latency without compromising the quality of rendered views of a 3D scene?

Our solution approach is algorithmic. As discussed above, where latency is likely to remain a challenge, we have chosen to solve this problem in an algorithmic way. Rather than re-architecting a 3D video streaming system for low latency, we design and implement a streaming algorithm that reduces the latency perceived by the user. This is achieved by proactively preparing the communication, which becomes visible by the following specific contributions.

- To reduce the user-perceived latency, we employ video prefetching, a streaming strategy where 3D scene-description data are fetched from the server before they are requested by the user.

- We optimize the bandwidth efficiency of prefetching by adapting the prefetching rate to user-navigation patterns, captured in a simple analytical model.

- To improve the rendering quality in face of time-varying delay or bandwidth, we propose an algorithm that implements an optimized rate allocation of the main and prefetching data, with the objective of smoothing the transitions between virtual views during navigation.

- We implement the proposed algorithm in a network simulator and demonstrate that it achieves a given target delay while maximizing the rendering quality.

Our proposal is based on a seminal paper [168], which was the first to emphasize the importance of prefetching in reducing the view-switching latency. To the best of our knowledge, we are the first to propose a complete interactive multiview 3D video streaming algorithm that achieves the goals set forth in [168].

This chapter is structured as follows. In Section 5.2, we provide an overview of dominant factors influencing the end-to-end system delays and survey related work.
in optimizations for reducing response latencies in 3D video streaming. In Section 5.3, we propose an algorithm for interactive 3D video streaming that achieves both a low interaction latency and a high rendering quality. In Section 5.4, we report on the algorithm implementation and performance, and Section 5.5 concludes the chapter.

5.2 Background and related work

5.2.1 Challenges in continuous 3D video scene-navigation

In this section, we first provide an overview of general factors influencing the system delay in interactive systems deployed over the Internet and the resulting challenges for a low-latency interaction. Then, we survey related work in interactive 3D video streaming. We conclude the section with a short summary, where we also discuss the specifics of interaction in 3D video streaming systems.

To illustrate the challenges in interactive 3D video scene-navigation, we use an example of a hypothetical 3D video streaming service deployed in the Internet, as illustrated in Figure 5.1. We assume that the 3D scene is captured with a multi-camera system. As in Chapter 3, the 3D data-representation consists of texture and depth layers that are compressed and stored at a server. Users can view the scene from different viewpoints and change the viewpoint interactively. Effectively, this represents a multiview 3D VoD-service [169], enhanced with the virtual-view rendering functionality. Even if the 3D scene is not originally captured from a requested viewpoint, the service synthesizes a virtual view for the user (Fig. 1.2). Similar to Chapter 4, we assume that the service supports a continuous view rendering along the user-selected navigation trajectory. An autostereoscopic display is not required for this service. Since the virtual views can be rendered on standard displays, this interactive service can even be deployed on some of today’s streaming receivers.

A. End-to-end system delay challenge

The end-to-end system delay and thus, user-experienced interaction latency, is the time interval between a user’s navigation request (small arrows to the left) and the moment a virtual view is rendered according to that request (large arrows via a different path in Fig. 5.1). The end-to-end system delay represents the total system delay, including both the upstream delays (for the propagation of a
navigation request to the sender) and the downstream delays (for the transmission of requested scene-description data to the receiver). In Internet video streaming, the system delay is a sum of service delays and network-transport delays. Table 5.1 summarizes various sources of delay, that would be present in an interactive 3D video streaming service implemented according to current service-implementation practices. For clarity, in Table 5.1, we indicate if the impact of a particular delay source will appear in upstream or downstream direction, or both. Let us now briefly discuss common sources of delay.

- **Service delay.** The service delay includes processing delays at each service component, including the receiver. At the receiver side, the decoding of texture and depth streams and rendering of virtual views are the main sources of delay. The components introducing delays at the service-provider side include: firewall processing to establish a secure service connection, processing for authentication and authorization of user requests and various optimiza-
5. Interactive Low-Latency 3D Video Streaming

The exact service delay depends on the service architecture and the total load in the system. Recent analyses of large-scale deployments of Internet streaming services suggest that the response latency is a secondary objective of service design [170, 163, 164]. In these systems, user requests are typically scheduled such that all servers are equally loaded, which provides cost savings and robustness against bursty request arrivals. Due to service provider’s focus on server load-balancing as the primary optimization objective, the response latency may be large [163, 164, 165]. Further, due to a dependence on system load, the response latency also varies over time [166].

- **Network-transport delay.** The second large contribution to end-to-end system delay is the network-transport delay. This delay is introduced both in the upstream direction during request routing and in the downstream direction during streaming (Fig. 5.1). Its main component is the RTT, which is commonly approximated as a sum of the transmission delay of the connection and the queueing delay in the network [35]. In congestion-controlled networks, the RTTs can be large and may vary by orders of magnitude. This is especially pronounced in wireless networks where RTTs on the order of seconds are not uncommon [171, 172, 173, 167]. Importantly for 3D video services, a single operation such as connection establishment by the transport protocol, may introduce a delay of 2–4 RTTs in each interactive request [174, 175].

In this chapter, we focus on the latency challenges posed to interactive 3D video streaming caused by the sum of delays at the service and network layers. More specifically, we assume that all sources of delay listed in Table 5.1 are active in the system, with the exception of receiver decoding and rendering delays.

Summarizing, the sum of delays at the service and network layers often leads to time-varying response latencies in the order of several seconds [167]. Although delay-optimized service and network architectures may mitigate this, such architectures remain hypothetical in view of the current trends [167].

**B. Quality-variation challenge**

To support the sense of immersion during a 3D-scene navigation, viewpoint changes need to be visually smooth. In this context, smoothness of navigation refers to
transitions between different virtual views rendered during navigation. Visible artifacts or quality changes during view transitions have a negative effect on immersive experience. An example is the “flicker” artifact, known to cause viewing discomfort in image-based navigation of static scenes [12]. An interactive 3D video streaming system should ensure gradual transitions between views and avoid such artifacts. However, ensuring smooth view transitions may be difficult for general multiview 3D video scenes, particularly when captured with insufficient sampling. As our work in Chapter 4 suggests, the quality of virtual views rendered from decompressed texture and depth streams needs to be carefully optimized over time. Our results in Chapter 4 demonstrate this for a sequence of virtual views rendered from a single viewpoint. When a 3D scene is rendered from multiple viewpoints over time, the quality-control problem is yet more difficult. Namely, the rendering quality of a 3D scene may be strongly viewpoint-dependent. We illustrate this with an experiment. We simulate an interactive user that requests a sequence of virtual views of the “Ballet” sequence [62]. We transmit a sequence of 40 virtual frames 20 times in succession, thus giving a total navigation time of 32 s. The viewpoints are selected such that the quality issue can be clearly demonstrated. Specifically, we assume that the user selects viewpoints that coincide with original camera locations ($Cam^{(1)}$, $Cam^{(2)}$, $Cam^{(3)}$ and $Cam^{(5)}$). We then use the rendering algorithm from Section 2.2.3 to render each of these virtual views using a pair of neighboring cameras ($Cam^{(0)}$ and $Cam^{(2)}$ are used to render the virtual view coinciding with $Cam^{(1)}$, etc.). In this way, we effectively reduce the scene-

**Figure 5.2:** Quality of 3D video rendering is view-dependent. Each virtual view is one GOP, PSNR is averaged per GOP; all views are virtual and in-between two references.
sampling rate. The quality variation during viewpoint transitions is illustrated in Figure 5.2. The coding rate of texture and depth layers and the GOP structure are the same for all views rendered in this experiment. These results demonstrate that the R-D functions of virtual views in a large multi-camera system have very different characteristics.

To better understand the quality-variation problem, in Figure 5.3 we provide an overview of different quality-control strategies in interactive 3D video streaming. For simplicity, we consider a scene description consisting of three texture and depth streams, depicted in Figure 5.3 (a). As the user interactively switches his viewpoint over time, different sets of four streams (two textures and two depths) are transmitted for decoding and rendering. The quality of rendered views depends on two types of distortion: the rendering distortion of the employed rendering algorithm and the coding distortion of the two textures and their depth maps. The rendering distortion depends on the scene complexity - in particular, the percentage of occluded pixels between the views and the algorithm’s occlusion-handling capability [65]. The coding distortions of texture and depth streams mainly depend on the effectiveness of the employed coding and rate control algorithms. As we show in Chapter 4, an effective control of the streaming rate crucially depends on the ability to optimize the coding rates of texture and depth streams. In an interactive streaming system, this optimization can be implemented using different strategies illustrated in Figures 5.3 (b)-(d).

1. **Independent rate control (Fig. 5.3 (b)).** The streaming rates of the texture streams can be optimized independently from the depth streams. Such an optimization is appropriate for simple streaming scenarios where the textures are displayed directly, or when view rendering is performed in a very narrow viewing range around the original viewpoints (e.g., for rendering on certain types of stereoscopic displays [11]).

2. **Joint rate control (Fig. 5.3 (c)).** A better optimization in the R-D sense is to jointly consider the compression of the texture and depth streams [176, 72]. The optimization method in [176] selects a single joint operating R-D point for a set of texture and depth streams. In particular, this method follows a two-step optimization process. First, for each texture and depth stream, the method renders virtual views corresponding to the positions of all other viewpoints in the set and computes distortions between the rendered and the original views. Second, it repeats this rendering for all available quantizer
Figure 5.3: Streaming rate control for interactive view switching in multiview 3D video streaming.
values and selects as the optimal quantizer the one that achieves minimum average distortion across all rendered views in the set.

3. **View-specific joint rate control (Fig. 5.3 (d)).** In Chapter 4, we have presented a joint texture-depth rate optimization that maximizes the quality of a single virtual view. In particular, we show that by using R-D functions of virtual views, we can significantly improve streaming performance in the R-D sense. This optimization is view-specific, since it requires to construct a separate R-D function for each viewpoint.

The choice of an appropriate optimization strategy for interactive multiview streaming needs to consider the following important aspects. Due to the lack of a joint texture-depth rate allocation, the independent rate control will likely lead to quality variations similar to those in Figure 5.2. The joint texture-depth rate control across a set of views [176] is more appropriate to our problem setting, due to its consideration of rendering quality across multiple views. A plausible solution strategy is to extend this coding rate optimization to streaming scenarios. However, a limitation of the method in [176] is its inability to optimize the quality of a given virtual view and smooth the transitions when switching views. Namely, this method focuses on general non-interactive scenarios where the virtual views to render are unknown during encoding, such that the averaging heuristic may indeed be appropriate for selecting an optimized operating point. Therefore, the accuracy of selecting an optimized operating point for a given virtual view will be limited. In contrast, the view-specific rate control (Fig. 5.3 (d)) allows to accurately select an optimized operating point for a single virtual view. When multiple views are rendered during a session, this optimization needs to be extended to account for smooth quality changes during view transitions. This is the general optimization approach we take and further develop in the remainder of this chapter.

### C. Efficiency challenge

A continuous 3D-scene navigation needs to be ensured, despite the time-varying available bandwidth in the Internet. The playout interruptions and rebuffering events should be avoided during navigation. Further, to support a low-latency interaction, the buffering delays should be minimized. This challenge is addressed in Chapter 4 for the single-viewpoint case. In this chapter, we focus on a more complex scenario of jointly optimizing response latency and bandwidth efficiency.
In more detail, the challenges addressed in this chapter differ from those in Chapter 4 in two important aspects. First, the system delay challenge is not considered in the design for bandwidth adaptation in Chapter 4. Second, the quality variation when rendering different virtual views of the same scene over time is a challenge specific to interactive scene navigation, which is the interaction scenario assumed in this chapter. In contrast, in Chapter 4, we have assumed that the user does not change his viewpoint during the session, such that we adaptively control the quality of a single virtual view over time. However, in this chapter, our bandwidth adaptation has to incorporate a solution for the more complex case of rendering multiple virtual views over time.

5.2.2 Related work

Although 3D video streaming systems are a nascent research area, the interactivity aspect has already received attention. The work in [168] is the first to identify response latency as a challenge in interactive multiview streaming. This work proposes prefetching to reduce view-switching latency and optimizes the coding rate allocation based on the user position. However, this multiview system does not include a virtual-view rendering functionality such that the interaction is limited to requesting one of the original camera streams. As we show in this chapter and in Chapter 4, a 3D video streaming system requires a different optimization compared to the conventional 2D video streaming. Similar ideas are considered in [177] to enhance user experience when viewing static 3D objects represented as 3D wireframe mesh models with mapped textures. However, this system does not consider a more complex scenario of dynamic 3D-scene viewing and adaptation to the bandwidth available in a congestion-controlled network. More recently, [178] and [179] propose codec optimizations for interactive streaming systems. The focus of these proposals is rate allocation at a macroblock-level. The quality of rendered virtual views is considered by weighting the macroblock-encoding quality proportionally to the macroblock’s importance for the rendered view. Similarly, the work in [180] considers a simplified scenario without virtual-view rendering and optimizes the inter-view prediction of the codec to achieve the desired trade-off between the storage space and the streaming rate. However, none of these proposals detail on the transmission aspects or propose a specific video streaming algorithm.
5. Interactive Low-Latency 3D Video Streaming

5.2.3 Summary

A low-latency navigation of a 3D-scene representation stored or generated in real-time on a remote server is challenging in the best-effort Internet. Today’s end-to-end system delays are often large and the trends in network design suggest the situation is unlikely to change in the future [167]. As a result, a 3D video streaming system needs to include algorithmic optimizations to ensure a continuous scene navigation despite the large and time-varying system delays.

Such an algorithmic solution needs to take into account that the user-interactive service model of 3D video streaming systems is different from the model of today’s conventional 2D video systems. First, different from today’s VoD systems that support a linear streaming with occasional “trick play” interactions (pause, fast forward, rewind, etc.) [148], the 3D video systems can be assumed to operate in the interactive mode most of the time. Further, unlike today’s low-latency video communications services [137], the 3D video system must additionally support a low-latency exploration of a remote 3D scene. For example, the user may judge a rebuffering event during a linear viewing differently than a delayed 3D-scene update triggered by his own interactive request. Finally, in contrast to today’s Internet services for Web and video content, a quality optimization is required to ensure that the view transitions are smooth. More generally, we assume that optimization objectives in 3D video systems need to be different not only semantically, but also perceptually.

Our assumption in this chapter is that algorithmic optimizations in 3D video streaming systems need to account for these differences. However, state-of-the-art algorithms fail to optimize for such an objective. To the best of our knowledge, we are the first to propose a complete algorithm that achieves a low interaction latency and an optimized rendering quality in 3D video streaming systems with large delay and time-varying available bandwidth.

5.3 Proposed algorithm

In this section, we propose a 3D video streaming algorithm that addresses the above challenges in a single framework. We first provide a high-level description of the algorithm, followed by a detailed presentation of the algorithm steps. The complete algorithm, which is the original contribution in this chapter, is given in Section 5.3.4.
5.3.1 Conceptual description

Conceptually, our algorithmic proposal is as follows.

- To reduce the user-perceived latency during navigation, we employ video prefetching, a streaming strategy where the texture and depth streams needed for rendering are fetched from the server before they are requested by the user. Since prefetching consumes a portion of the available bandwidth, the prefetching rate needs to be allocated efficiently. We optimize the bandwidth efficiency of prefetching by adapting the prefetching rate to user-navigation patterns. An analytical model is used to represent these patterns.

- To ensure a continuous streaming and a high streaming quality despite the network dynamics, we further adapt the total 3D video streaming rate (a sum of the rates of the “main” and “prefetching” streams) to a congestion-controlled network transmission rate.

- The proposed adaptations use an optimized rate allocation of the main and prefetching streams, with the objective of smoothing the transitions between virtual views during navigation. In this step, we perform an optimized selection of operational R-D points for the texture and depth streams, based on the algorithm presented in Chapter 4.

A high-level view of the proposed algorithmic framework is shown in Figure 5.4. We use the layered 3D video representation from Chapter 3 and the view-rendering algorithm from Section 2.2.3. For brevity, we refer to texture and depth streams as reference streams. When a user requests a virtual view, the 3D video server transmits the compressed main reference streams required for rendering. The receiver decodes the streams and renders the scene from the requested viewpoint. By fetching a selection of the reference streams before they are requested, we can serve future interactive requests by rendering the scene from the prefetching reference streams available in a local streaming buffer. In this way, we avoid a potentially large system delay and ensure a low response latency. Although prefetching is a well-known performance enhancement for large-delay systems, 3D video systems are specific due to the user-interaction aspect.

In the remainder of this section, present the details of the proposed algorithm. In Section 5.3.3, we describe the particular technique used in our algorithm, referred to as user-adaptive video prefetching. To increase bandwidth efficiency of
prefetching, the algorithm tracks the navigation patterns according to a 3D user-interaction model (Section 5.3.2) and dynamically allocates the streaming rates to the main and prefetching streams (Section 5.3.4). Finally, similar to Chapter 4, the adaptive streaming control component tracks the bandwidth variation and adapts the streaming rate (Fig. 5.4). In the sequel, we present the proposed algorithm and its main components in detail. In Figure 5.5, we provide a roadmap to the remainder of this section in order to help the reader quickly locate our main contribution in this chapter.
5.3.2 User-interaction model for multiview 3D video navigation

An ideal 3D video streaming algorithm would prefetch only as much data as needed to satisfy the target response-latency, without consuming additional bandwidth. For example, with a parametric model of navigation patterns measured across different users, the system could instantiate the model for a particular user and optimize prefetching performance. However, due to the lack of large-scale deployments, such models are currently unavailable for 3D video systems, so that we have to construct a synthetic model. This approach was successfully applied to improve the interactive performance in similar systems [168, 178].

Our starting point is a modeling guideline from Web browsing, a related research area where techniques for minimizing web-page latencies have been studied earlier [174]. A user-interaction model constructed for latency-minimization purposes must capture three dependencies [181]: (1) application-dependence, (2) user-interface-dependence and (3) user-dependence.

As discussed in Section 5.1, our application scenario is a multiview exploration of a remote 3D scene. A user changes the viewpoint interactively, while the system synthesizes virtual views in real-time, according to the actual navigation trajectory.

An interactive navigation of a 3D scene can be implemented using a real-time feedback channel, as in our layered streaming model in Chapter 3. The user interface for scene navigation in our streaming prototype is implemented using standard PC interface devices (Chapter 3). The user initiates a view-switching request by moving the mouse pointer inside of his local video display-window. The pointer position is converted from the display-window coordinates to the coordinates of reference cameras in a 3D-scene coordinate system. A virtual view with the specified scene coordinates is then rendered and projected back into window coordinates prior to display. The Eq. (3.1) in Section 3.4.2 represents this 3D mapping between the display and scene coordinates.

Since our rendering algorithm synthesizes virtual views along a horizontal chain of cameras, only the “x” display-window coordinate is relevant for navigation. Correspondingly, each of the reference cameras $Cam^{(1)}, \ldots, Cam^{(N-1)}$ can be represented with a display coordinate of its center $x_i^{(0)}, \ldots, x_i^{(N-1)}$. Thus, we can omit the superscript and refer to the display coordinate as $x_i$, while assuming that an original view at position $x_i$ is a special case of a virtual view where only one stream is transmitted.
The remaining aspect of the model is user-dependence. For our purposes, the model of user-dependence refers to a distribution of user requests over time. Instead of assuming a global theoretical model for this distribution, we assume a local model in the form of navigation velocity $v_{\text{nav}}$, as in the related area of Distributed Virtual Environments [182]. In our case of horizontal navigation, the navigation velocity lends itself to the following formulation, as illustrated in Figure 5.6. Assuming that at consecutive time instants $t(k-1)$ and $t(k)$ the user requests two different views $x_i(k-1)$ and $x_i(k)$, respectively, the instantaneous navigation velocity can be written as:

$$v_{\text{nav}}(k) = (x_i(k) - x_i(k-1))/(t(k) - t(k-1)).$$  \hspace{1cm} (5.1)

Using Eq. (5.1) and Eq. (3.1), we maintain a one-to-one mapping between the user-navigation patterns in time and display coordinates, and the camera coordinates in scene space. Equivalently, we can describe an arbitrary navigation pattern over time, in the part of the 3D scene delimited by the original cameras.

### 5.3.3 User-adaptive video prefetching

Assuming an end-to-end system delay $d(k)$ at time instant $t(k)$, a requested view $x_i(k)$ will be rendered as time-shifted $x_i(k+d(k))$, where $d(k)$ is the sum of delays at the service and network layers (as described in Section 5.2.1). With prefetching, we fetch the reference streams required to render the view $x_i$ such that it can be rendered as $x_i(k+d_{\text{tar}})$, where $d_{\text{tar}}$ is the target response-latency when navigating a multiview 3D scene (e.g., set by the 3D-service provider). Due to a general unpredictability of user-navigation patterns, the prefetched data may actually never be displayed. Since the prefetching reduces the bandwidth available
for transmission of the main streaming data, fetching unnecessary data could significantly reduce the average video quality during a session. Although such events cannot be completely avoided, we can minimize their occurrence by adapting to user-navigation patterns. To this end, our prefetching uses a running estimate of the navigation velocity \( v_{\text{nav}}(k) \) and employs it to predict a viewpoint \( x_i \) that will be requested in the future. Specifically, according to Eq. (5.1) and assuming the current view position \( x_i(k) \), the one-step-ahead predicted view position \( \hat{x}_i(k+1) \) can be written as:

\[
\hat{x}_i(k+1) = v_{\text{nav}}(k) \cdot (t(k+1) - t(k)) + x_i(k).
\] (5.2)

Instead of directly using the instantaneous navigation velocity \( v_{\text{nav}} \), we use its smoothed estimate, based on an Exponentially Weighted Moving Average filtering (EWMA):

\[
\hat{v}_{\text{nav}}(k) = \begin{cases} 
0, & \text{if } \forall v_{\text{nav}}(k-i) = 0, i = 1 \ldots m \\
\frac{1-\alpha}{1-\alpha^n} \sum_{i=1}^{n} \alpha^{n-i} v_{\text{nav}}(k-n+i), & \text{otherwise,}
\end{cases}
\] (5.3)

where \( \alpha \) is the smoothing factor, \( n \) is the size of the smoothing window and \( m \) is the number of consecutive zero-valued velocity estimates after which we reset the estimator. The EWMA-smoothed estimate is shown to give most accurate prediction of 3D position vectors in [182]. Differently from [182], we apply it to our estimates of the velocity \( \hat{v}_{\text{nav}}(k) \).

Intuitively, the smoothed average of \( v_{\text{nav}} \) captures the average navigation intensity in a time interval \([t(k-n+1), t(k)]\). A consecutive sequence of non-zero values of \( v_{\text{nav}} \) suggests that the user is actively exploring the 3D scene and our prediction emphasizes that the user will continue doing so, at least for one prediction step ahead into the future \( t(k+1) \). Likewise, consecutive zero-values of \( v_{\text{nav}} \) suggest that the user settles at the current viewpoint and is likely to remain in the non-interactive, linear viewing mode.

### 5.3.4 Complete algorithm

The spatial quality, or its dual – the distortion, of a virtual view rendered from decompressed reference streams can be expressed using the MSE metric, according to Eq. (4.4). Using this distortion metric, an operational R-D point can be selected for each virtual view independently. However, the R-D functions of
virtual views exhibit a strong viewpoint-dependence (Fig. 5.2). As a result, an independent quality optimization may lead to visible quality changes during view transitions and negatively affect the overall perceptual quality. Further, any such quality optimization must be performed under a limited and time-varying available bandwidth, typical for the best-effort Internet.

We therefore propose an algorithm that jointly optimizes the quality of virtual views rendered during an interactive streaming session. The algorithm achieves an efficient bandwidth utilization and smooth view transitions as follows. First, it schedules a transmission of the main and prefetching streams such that the target response-latency $d_{\text{tar}}$ is satisfied. In this step, we employ the user-adaptive prefetching from Section 5.3.3. Second, the algorithm controls the aggregate streaming rate such that the receiver buffer does not underflow due to available-bandwidth variations. We rely on the adaptation algorithm from Chapter 4 and extend it to include both the main and the prefetching reference streams. Third, the algorithm performs a selection of operational R-D points for the reference streams with the objective of smoothing the transitions between virtual views during navigation. Given multiple versions of the texture and depth reference streams and the R-D functions of virtual views generated as in Chapter 4, the proposed algorithm consists of the following steps.

Algorithm for quality-optimized adaptive prefetching

**Step 1: Estimating the prefetching streams.**

Given the current virtual view position $x_i(k)$ in user display coordinates, we compute a predicted view $\hat{x}_i(k+1)$ according to Eq. (5.2). In this step, we use an estimate of navigation velocity $\hat{v}_{\text{nav}}(k)$ obtained as a result of EWMA-smoothing in Eq. (5.3). The predicted view $\hat{x}_i(k+1)$ is then supplied to Eq. (3.1), to determine the required prefetching texture and depth streams.

**Step 2: Scheduling the reference streams to satisfy $d_{\text{tar}}$.**

The current virtual view $x_i(k)$ and the one-step-ahead predicted virtual view $\hat{x}_i(k+1)$ are both transmitted in the same time slot $k$. The duration of time slots $[k, k+1]$ should be short enough in order to adapt to different values of the target response-latency $d_{\text{tar}}$. In practice, we expect that latencies of up to 1 s will offer a satisfactory navigation experience [183]. Correspondingly, a reasonable setting for the duration of the time slot $[k, k+1]$ can be in the order of hundreds of milliseconds.
Step 3: Computing the aggregate rate for adaptive streaming.

Similar to Section 4.4.1, we adapt the streaming rate to avoid starvation of the receiver playout buffer. In our case of transmitting the main and prefetching reference streams in the same slot $k$, a modified leaky bucket control can be expressed as follows. Assuming that in each time slot $k$, $R(k) = R_{\text{main}}(k) + R_{\text{pref}}(k)$ bits are added to the bucket and $r(k)$ bits are read and removed from the bucket, the condition to prevent bucket overflow $L(k) > b$ can be expressed as:

$$L(k) = \max(0, L(k-1) + R_{\text{main}}(k) + R_{\text{pref}}(k) - r(k)).$$  \hfill (5.4)

Correspondingly, in each time slot $k$, our algorithm adapts the rate of the main streams $R_{\text{main}}(k)$ and the rate of prefetching streams $R_{\text{pref}}(k)$ to ensure that the receiver buffer does not underflow. The transmission rate $r(k)$ is estimated using the algorithm in Section 4.4.1.

Step 4: Smoothing the quality of virtual views.

We use the Algorithm 3 (Section 4.3.3) to obtain a smooth quality variation across the rendered virtual views. This algorithm computes a joint texture-depth rate allocation for both the main and the prefetching streams, according to the R-D-theory criterion of minimizing the distortion variation. In case of non-zero velocity estimates, Algorithm 3 is applied to minimize the distortion variation among $K$ virtual views that will be rendered in future time slots $k, \ldots, k+K-1$. For this prediction, we assume velocity estimates $\hat{v}_{\text{nav}}(k) = \hat{v}_{\text{nav}}(k+1) = \cdots = \hat{v}_{\text{nav}}(k+K-1)$ and bandwidth estimates $r(k-1) = r(k) = \cdots = r(k+K-1)$. In case of linear viewing, i.e., a zero-valued velocity estimate, Algorithm 3 performs a temporal quality optimization as described in Section 4.3.3.

Quality control dynamics. We note that our assumptions on the constancy of velocity and bandwidth estimates may not be satisfied in scenarios characterized by highly dynamic navigation patterns or bandwidth variations. However, such scenarios are not a limitation for our algorithm, which we justify as follows. In case of an abrupt change in velocity or available bandwidth at time instant $t(k+1)$ compared to $t(k)$, the algorithm computes a new quality-optimized allocation at time $t(k+1)$, using the most recent velocity and available-bandwidth estimates. This leaves us with the problem of an abrupt change within slot $k$. The duration
of slots $k$ in our algorithm implementation is 400 ms. At such short timescales, the probabilities of large variations of available bandwidth and navigation patterns are low, as shown in [156] and [182], respectively. In case of such an event, a streaming system can skip virtual frames that cannot be rendered from the buffered data or employ efficient concealment techniques [137].

5.4 Implementation and performance evaluation

In this section, we implement the proposed algorithm for interactive low-latency 3D video streaming and perform experiments to demonstrate that it achieves the properties stated in Section 5.3. Our implementation extends the algorithms and components developed in Chapter 3 and Chapter 4. Streaming experiments are performed according to the evaluation methodology designed in Section 4.4.2.

5.4.1 Algorithm implementation

The quality-optimization algorithm proposed in Section 5.3.4 assumes the availability of multiple versions of the texture and depth reference streams and the combined R-D functions of virtual views. We apply the multi-version encoding method described in Section 4.4.1 and extend it to better suit interactive 3D VoD services introduced in Section 5.1. The proposed extensions are service-specific coding techniques that can be used to optimize the efficiency of real service deployments.

A. Camera configuration

The Step 1 of the proposed algorithm determines the required set of prefetching reference streams using an estimate of the navigation velocity $\hat{v}_{\text{nav}}$ (Section 5.3.4). Depending on the estimated velocity and the configuration of the original cameras, there are two implementation options, illustrated in Figure 5.7. We refer to the camera configurations depicted in this figure as adjacent-camera (Fig. 5.7 (a)) and skip-camera (Fig. 5.7 (b)). The adjacent-camera is a common configuration where a virtual view is rendered from the two adjacent texture streams (one at the left and one at the right of the requested viewpoint) and their depth streams. The skip-camera effectively reduces the scene-sampling rate in the navigation direction, which may reduce the rendering quality. At the same time, as Figure 5.7 (b) illustrates, the skip-camera also allows to reduce the number of required original
Figure 5.7: Navigation velocities $v_{nav1} < v_{nav2}$ and camera configurations. (a) Adjacent-camera configuration. (b) Skip-camera configuration.

views. This property is useful in an interactive multiview 3D streaming system, e.g., when the actual navigation speed is higher than our estimate. We detail on this property in the sequel.

The skip-camera may be used to increase robustness of system implementations in case of large prediction errors in Eq. (5.2). We use the navigation scenarios in Figure 5.7 as an example. A robust implementation could prefetch the data for view $x_1$ in the slot $[k, k + 1]$, to ensure that the correct view can be rendered in case of an erroneous prediction. For the adjacent camera, the range of velocities $[0, v_{nav1}]$ would require to transmit 6 reference streams in slot $[k, k + 1]$: $(Cam^{(i+1)}, Depth^{(i+1)}, Cam^{(i+2)}, Depth^{(i+2)})$ for predicted view $\hat{x}_2$ and $(Cam^{(i)}, Depth^{(i)})$ required to render view $x_1$ in case of a prediction error. For higher velocities $[v_{nav1}, v_{nav2}]$, 8 reference streams will be transmitted: the predicted view $\hat{x}_3$ requires to transmit $(Cam^{(i+2)}, Depth^{(i+2)}, Cam^{(i+3)}, Depth^{(i+3)})$ and the view $x_1$ requires $(Cam^{(i)}, Depth^{(i)}, Cam^{(i+1)}, Depth^{(i+1)})$. In contrast, the skip-camera configuration would render the view $\hat{x}_3$ by skipping the reference streams $(Cam^{(i+2)}, Depth^{(i+2)})$ and rendering with $(Cam^{(i+3)}, Depth^{(i+3)})$ instead. As a result, it would require to transmit 6 reference streams, resulting in a lower aggregate rate requirement and potentially a higher rendering quality.

This suggests that the skip-camera configuration may be an efficient implementation option for interactive 3D video services. Correspondingly, we encode multiple 3D video versions in both camera configurations and use them to evaluate the algorithm performance. For the adjacent-camera configuration, we follow the approach described in Section 4.3.2 to determine the joint operating point for
Algorithm 4: Extract a Set of R-D Optimized 3D Video Versions

Data: N textures and N depths corresponding to N viewpoints

foreach Viewpoint \( V^{(n/2)} \) to \( V^{((N-2)/2)} \) do

- Render virtual \( V_{o-adj}^{(n/2)} \) from raw textures and depths in adjacent-camera configuration;

foreach Texture quantizer \( Q_t = Q_{min} \) to \( Q_{max} \) do

- Compress two textures at \( Q_t \);

foreach Depth quantizer \( Q_d = Q_{min} \) to \( Q_{max} \) do

- Compress two depths at \( Q_d \);

- Render virtual \( V_{c-adj}^{(n/2)} \) from decompressed textures and depths;

- Compute \( D_{adj} \) (MSE) in \( V_{c-adj}^{(n/2)} \) compared to \( V_{o-adj}^{(n/2)} \);

Assign \( D_{adj} \) to RDFunction[\( V^{(n/2)},R_t,R_d \)];

\( R_{min} = R_{tmin} + R_{dmin} \);

\( R_{max} = R_{tmax} + R_{dmax} \);

\( \text{samplingStep} = \frac{\log(R_{max}) - \log(R_{min})}{s} \);

foreach \( i = 1 \) to \( s \) do

- \( R_{ti} = 10^{(\log(R_{min})+(i-1)\cdot\text{samplingStep})} \);

- \( D_{adj}^{min}(R_{ti},R_{di}) = \arg \min_{R_t,R_d} \text{RDFunction}[V^{n/2},R_t,R_d] \text{s.t.} R_t + R_d \leq R_{di} \);

Store the 3D video version corresponding to point \( (R_{ti},R_{di}) \);

encoding. Specifically, the joint operating point \( (R_t, R_d) \) to encode the textures \( Cam^{(i)} \) and \( Cam^{(i+1)} \) and their depth streams \( Depth^{(i)} \) and \( Depth^{(i+1)} \) is determined by iteratively evaluating Eq. (4.4). For the skip-camera configuration, we use a modified procedure as follows. Due to a larger camera baseline of this configuration, for each pair of cameras there exists an original camera between them. This camera can be readily employed as a reference in Eq. (4.4). Correspondingly, we use \( Cam^{(i+1)} \) as the reference to encode \( Cam^{(i)} \) and \( Cam^{(i+2)} \) and their depth streams \( Depth^{(i)} \) and \( Depth^{(i+2)} \).

B. R-D optimized version extraction

The multiple encoded 3D video versions need to be stored at the server and scheduled for transmission as in Step 2 of the proposed algorithm (Section 5.3.4). The number of versions to encode, store and schedule may be large for a 3D scene that covers a very large area, or a 3D scene that is sampled with high density in the camera plane. This may impose significant storage and management costs for
Figure 5.8: R-D optimized version extraction: in the set of all available rate points, 9 optimized rate points are extracted as in Algorithm 4.

We describe an optimization that reduces these costs by selecting a limited number of 3D video versions to store. This optimization is performed as follows. We select a fixed number of rate points $R^i$ (total rate of four reference streams – textures and depths) that uniformly sample the range $[\log R_{\min}, \log R_{\max}]$ achievable by the quantizers in our encoder $[Q_{\max}, Q_{\min}]$. We then encode and store only the texture and depth streams that achieve minimum distortion of the rendered virtual view under the rate constraint $R^t_i + R^d_i \leq R^i$. As in Section 4.3.2, the $R^t_i$ and $R^d_i$ are the combined rates of the two texture streams and the two depth streams, respectively. The pseudocode is given in Algorithm 4. In this example, we extract a set of $s$ R-D optimized 3D video versions in the adjacent-camera configuration. Figure 5.8 illustrates the result for $s = 9$ versions. The optimized 3D video versions for the skip-camera configuration can be computed with the same algorithm. The trade-off in this service optimization is a reduced allocation accuracy compared to the theoretical case of employing all achievable versions. We quantify this trade-off as a part of our performance evaluation.
5. Interactive Low-Latency 3D Video Streaming

5.4.2 Performance evaluation

A. Simulation setup

To evaluate performance of the proposed algorithm, we employ the methodology designed in Section 4.4.2. The trace-driven simulation of video transmission [157] enables experiments with actual 3D video sequences, rendering and coding algorithms. In addition, the use of a packet-level network simulation (ns-2) allows a realistic evaluation of the algorithm performance in face of time-varying delay, available bandwidth and real-world implementations of transport protocols.

We perform a set of experiments dealing with typical situations involved with low-latency interactive multiview 3D video streaming, that demonstrate the main properties of the proposed algorithm, as stated in Section 5.3. Specifically, we show that a 3D-scene navigation using the proposed algorithm is continuous and with low latency, despite the time-varying delay and available bandwidth. We also demonstrate the algorithm’s efficiency achieved by adapting to user-navigation patterns and the available bandwidth. A separate set of experiments is performed to demonstrate that the algorithm enables smooth view transitions. These experiments include an analysis of camera configurations and a visual evaluation of the rendering quality. Simulation parameters common to different experiments are summarized in Table 5.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiview sequence</td>
<td>“Ballet”, 8 original cameras</td>
</tr>
<tr>
<td>Frame rate</td>
<td>25 fps</td>
</tr>
<tr>
<td>Frame resolution</td>
<td>1024×768 pixels</td>
</tr>
<tr>
<td>GOP size</td>
<td>10</td>
</tr>
<tr>
<td>Number of versions ( s )</td>
<td>9</td>
</tr>
<tr>
<td>Quality-optimization interval (Table 4.1)</td>
<td>1.2 s</td>
</tr>
<tr>
<td>Congestion-control algorithm</td>
<td>TFRC</td>
</tr>
<tr>
<td>Queue management</td>
<td>RED with ECN</td>
</tr>
<tr>
<td>Simulation duration</td>
<td>32 s</td>
</tr>
<tr>
<td>Target response-latency ( d_{\text{tar}} )</td>
<td>400 ms</td>
</tr>
<tr>
<td>EWMA smoothing window ( n )</td>
<td>3 (1.2 s)</td>
</tr>
<tr>
<td>EWMA smoothing factor ( \alpha )</td>
<td>0.8</td>
</tr>
<tr>
<td>Estimator reset ( m )</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.2: Parameters for the experimental evaluation.
B. Results

Figure 5.9 (a) shows a typical example of algorithm’s adaptation to user-navigation patterns. To better illustrate this aspect, we create a scenario where the available bandwidth is constant. The simulated user-navigation pattern consists of a period of linear viewing ($v_{nav} = 0$), followed by a navigation episode at velocity $v_{nav1}$ (Fig. 5.6). While the user is navigating the scene, the available bandwidth is shared between the main and the prefetching streams. As a result, the average rendering quality is lower than during linear viewing. This result is expected and explained by the need to prefetch a larger number of reference streams for the target response-latency $d_{tar}$. Importantly, our algorithm achieves a consistently low interaction latency during this period. As the navigation velocity decreases, our algorithm reduces the prefetching rate, thus allocating a higher rate to the main streams and improving the rendering quality. In this way, the algorithm effectively optimizes the balance between the interaction latency and rendering quality. Importantly, this optimization is user-adaptive. As a result, although wildly changing navigation velocities may cause fluctuations of rendering quality, our adaptation ensures that if the user-navigation patterns are smooth, the fluctuations of rendering quality will also be smooth.

In Figure 5.9 (b), we illustrate the algorithm’s ability to provide smooth view transitions despite the view-dependent quality variations inherent in a multiview scene. In this experiment, the available bandwidth is relatively stable. Our interactive 3D video navigation-session shares the network capacity with one long-duration TCP flow and one Web flow. We simulate a navigation at constant velocity $v_{nav1}$ (Fig. 5.6). Thus, in each slot $k$, a virtual view is requested that requires a different set of reference streams compared to previous slot $k - 1$. We select this scenario as the most challenging for our quality smoothing. Due to the large physical distance between the views rendered in consecutive time slots, the differences of R-D functions are visible in each slot. The requested views are positioned at regular intervals along an arc connecting the camera centers of $Cam(0), \ldots, Cam(5)$. Using the notation from Section 4.3.2, we refer to these views as $V^{(0/2)}, V^{(1/2)}, V^{(2/2)}, V^{(3/2)}, V^{(4/2)}$ and $V^{(5/2)}$. The algorithm schedules per-slot view transmissions as in Section 5.3.4. Figure 5.9 (b) shows the resulting quality profile over time. For comparison, in the same figure we plot the quality profile of an unoptimized transmission that allocates equal streaming rates to the main and prefetching views. It can be seen that the proposed algorithm
Figure 5.9: Algorithm performance - adaptation for adjacent-camera configuration: (a) Adaptation to user-navigation patterns; (b) Quality-smoothed view transitions; (c) Simultaneous adaptation to available-bandwidth variations and user-navigation patterns.
smooths the quality differences, while maintaining a high average quality. We note that the quality in this experiment is quantified according to the PSNR metric. Since the PSNR represents the average quality of rendered frames in one GOP, it may not always reflect the occurrence of rendering artifacts in individual regions of a frame, or between frames. Our additional subjective viewing tests confirm that the view transitions are smooth, without disturbing rendering artifacts.

Figure 5.9 (c) illustrates the performance of the proposed algorithm in the most complex scenario of simultaneously adapting to user-interaction patterns and the available bandwidth. The dynamics of the available bandwidth are simulated with a cross-traffic consisting of 3 additional 3D video streams, as described in Section 4.4.2. When the network is in steady state, we simulate a linear viewing ($v_{nav} = 0$). The cross-traffic is started at $t = 16$ s (GOP Index 40). The bandwidth drop due to the burst of cross-traffic is visible only after a few seconds, as the cross-traffic streams are going through a TFRC slow-start phase. This is visible in Figure 5.9 (c) as a marked quality drop at $t = 24$ s (GOP Index 60). At $t = 24$ s (GOP Index 60), we stop the cross-traffic streams. While the algorithm gradually increases quality, we start a navigation episode corresponding to $v_{nav} = v_{nav1}$ (Fig. 5.6). At this point, the algorithm starts to prefetch the reference streams, which is visible as the quality drop at $t = 27$ s (GOP Index 66) in Figure 5.9 (c). The navigation episode ends at $t = 30$ s (GOP Index 75), after which the algorithm stops prefetching. As a result of allocating more rate to the main streams, it achieves a higher rendering quality near the end of this simulation scenario.

Figure 5.10 illustrates the quality-adaptation performance for the skip-camera configuration.
configuration. The cross-traffic consists of 3 additional 3D video streams, as in Figure 5.9 (c). Consistent with the scenario in Figure 5.2, we assume that the user selects viewpoints that coincide with original camera locations (Cam\(^{(1)}\), Cam\(^{(2)}\), Cam\(^{(3)}\) and Cam\(^{(5)}\)). The results show that notable quality variations remain in the rendered views and that the quality smoothing is less effective than in the adjacent-camera configuration. This performance difference is consistent in the entire navigation session and for each individual virtual view. We explain this result as follows. The “Ballet” sequence used in our experiment has a large camera baseline and is particularly challenging for virtual-view rendering. In the adjacent-camera configuration, the rendering algorithm successfully resolves most disocclusions. For a yet larger baseline of the skip-camera configuration, the rendering algorithm is pushed to its maximum. Although the visual quality of synthesized views is acceptable, their objective quality level is significantly below the one achieved with the adjacent-camera configuration. Further, the adaptation range achievable by modifying the quantizer \(Q\) in the skip-camera configuration is significantly reduced. To better understand this, in Figure 5.11, we plot the R-D function of a virtual view corresponding to Cam\(^{(3)}\) in the original sequence. It can be seen that the achievable quality-adaptation range for the skip-camera configuration is limited to only about 1 dB.

In Figure 5.12, we additionally compare our streaming service-efficiency optimization that uses a limited number of 3D video versions to the “optimal” case of using all achievable versions. We perform this comparison for the adjacent-camera configuration. The results show that the proposed service-efficient encoding effectively smooths the quality variations, achieving a performance comparable to
that of the “optimal” allocation. In this experiment, the service-efficient encoding stays only 0.23 dB below the average quality achieved in the “optimal” case. In our view, this is experimental evidence for the existence of good engineering solutions with a quality very close to the optimal streaming case. In particular, the number of versions to store for a single virtual view is reduced from quadratic in the number of R-D points \((R_t \times R_d)\) to a constant. This has the following advantages in practice. First, storage cost is reduced, thereby reducing the total system cost of which the cost of storage is one major component [92]. Today’s state-of-the-art 2D streaming solutions deploy a similar approach to cost saving by storing a small number of encoded versions (typically, five different versions are stored [36]). In contrast to these solutions that do not consider video quality when selecting the versions to store, our method selects them such that the quality of rendered views is maximized. Second, computation cost is reduced as a result of the reduction in the number of R-D points to evaluate in every adaptation interval. This cost is important in practice, as an overly high computation cost may limit service capacity in terms of the number of concurrent users, or degrade user experience by causing load-dependent response latencies [166]. With our method, both cost reductions are achieved at a small penalty in video quality (0.23 dB). We therefore believe that this encoding method is useful for 3D-scene representations in general and for scenes that contain a large number of captured views in particular. Currently, we can only provide experimental evidence for this claim, by showing that the R-D function of virtual views for our test 3D scene is sufficiently densely sampled with nine selected rate points. A derivation of the optimal number of versions to store for a given 3D scene requires a theoretical underpinning that we lack at this stage.

In Figure 5.13, we visually evaluate the proposed adaptation with adjacent-camera and skip-camera configurations. For this experiment, we render a virtual view that coincides with the original camera location \(Cam^{(5)}\). This view is rendered from the reference streams corresponding to \(Cam^{(4)}\) and \(Cam^{(6)}\) for the adjacent-camera configuration and \(Cam^{(4)}\) and \(Cam^{(7)}\) for the skip-camera configuration. The visual inspection shows that the rendered frames have acceptable quality in both configurations. This holds in the entire range of coding-quality settings used in this chapter. Although the encoding introduces distortions in the reference streams, the view-rendering algorithm reduces their visibility in the rendered views. In the lowest-quality encodings of reference streams, blockiness artifacts become visible. This is an artifact of block-based MPEG coding in gen-
5. Interactive Low-Latency 3D Video Streaming

Figure 5.12: Service-efficiency optimization that uses a limited number of 3D video versions.

...eral and is not specific to our method. Although the skip-camera configuration leads to more disocclusions than the adjacent-camera configuration, the rendering algorithm resolves most of them such that they do not significantly degrade the visual quality. However, in lowest-quality encodings, we observe mosquito-noise-like artifacts around the object edges. This is a well-known artifact with 3D data-representations based on depth maps and is due to depth inaccuracies along the depth discontinuity edges [62]. From the compression standpoint, its effect may be mitigated by employing specialized depth codecs [72]. Nevertheless, these visual comparisons are experimental evidence of the usefulness of the proposed approach in real 3D video streaming systems.

5.5 Conclusions

Our focus in this chapter is the challenge of large and time-varying latencies in interactive 3D video streaming systems. Specifically, we focus on providing an algorithmic solution for reducing user-perceived latency without compromising the quality of the rendered views.

To this end, we have proposed and implemented an algorithm for interactive 3D video streaming that achieves a low interaction latency and a high rendering quality. We have demonstrated that the proposed algorithm achieves these properties using analysis and experiments with realistic network-transmission conditions and actual 3D video sequences, rendering and coding algorithms. Specifically, our contributions in this chapter are listed below.
Figure 5.13: Visual comparison of adaptation results, “Ballet” camera Cam\(^{(5)}\): (a) Ground truth (original view); (b) Adjacent-camera; (c) Skip-camera; (d) Edge artifacts in skip-camera configuration due to a high depth-compression ratio.
• 3D-scene navigation with low response latency. To reduce the user-perceived latency in 3D video systems with large end-to-end delays, our algorithm prefetches the reference texture and depth streams needed for rendering. This is achieved with a 3D user-interaction model and a scheduling strategy that allows to satisfy a predefined latency target. Further, by adapting to user-navigation patterns, the algorithm optimizes the bandwidth efficiency of prefetching. We have experimentally validated the correctness of the adaptation to user-navigation patterns.

• Quality-smoothed view transitions. The proposed algorithm smoothes the view-dependent quality variations inherent in a 3D multiview scene by applying an optimized rate allocation of the main and prefetching streams. This allocation is based on minimizing the distortion variation between rendered views and is achieved with a joint texture-depth R-D optimization. The experimental results demonstrate that the algorithm is capable of smoothing the view transitions in the PSNR sense. In addition, we provide visual evaluation to show that visually disturbing artifacts are avoided.

• Efficient utilization of the available bandwidth. To ensure an efficient bandwidth utilization, the algorithm further adapts the aggregate 3D streaming rate (the sum of the rates of the “main” and “prefetching” streams) to a congestion-controlled network transmission-rate. This adaptation is performed simultaneously with the adaptation to user-navigation patterns. Our experimental results show that the algorithm prevents buffer starvation and thus ensures a continuous navigation.

With respect to the overhead of the proposed prefetching solution, we conclude as follows. Since our algorithm adapts to a congestion-controlled connection, there is no network overhead. Instead, the price of prefetching is paid in the form of a reduced rendering quality during navigation episodes. Unfortunately, the current algorithm does not optimize this trade-off. For example, we only prefetch the data for one virtual view into the future, regardless of the actual system delay. As a result, the performance of our algorithm may suffer in the following scenarios. In case of underestimating the navigation velocity, the delay target may not be met, i.e., the perceived latency of rendering the requested virtual view may be larger than desired. Conversely, in case of overestimating the navigation velocity, the rendering quality may be unnecessarily reduced. Although our algorithm is
designed to minimize the impact of such events (Section 5.3.4), we cannot avoid this in the absolute sense. Moreover, it is difficult to quantify their impact on user satisfaction. Since the impact on user satisfaction is visible only during one time slot, it is very transient and hard to generalize this to the user satisfaction related to the entire session.

Our analysis of the impact of camera configuration on the rendering quality is preliminary. Despite a consistently worse performance of the skip-camera configuration in our experiments with large-baseline sequences, we note that the performance difference depends both on the scene complexity and on the rendering algorithm. Our visual inspection of the rendering results confirms the usefulness of the skip-camera configuration in practice. For this reason, we believe that the skip-camera configuration can be a useful optimization in interactive multiview systems, especially in view of the fact that such an optimization opportunity does not exist in single-view video prefetching. More generally, further research is needed on adaptive algorithms that simultaneously modify the compression ratio and the camera configuration to control the streaming rate. For example, such algorithms could dynamically compare the two configurations and select the one with lower distortion at the same prefetching rate.

Our presented work is the first step towards efficient interactive 3D video streaming. We are convinced that the concepts and algorithmic components proposed in this chapter will be useful as building blocks to design future 3D video systems. More generally, we conjecture that they can be used in other interactive services that aim to maximize the perceptual service quality. Further work is needed on view-dependent models of R-D functions of virtual views, since the functions used in our work are view-specific. In addition, availability of such models could potentially reduce the complexity of constructing the R-D functions in practice. For large or densely sampled 3D scenes, the construction of R-D functions may require a very large number of encoding and rendering iterations [177]. In addition, a model would allow a relative comparison of camera configurations, in contrast to the MSE/PSNR metric that requires a common reference. Recently, the first analytical models have been derived for use in R-D optimized video coding [150, 184, 185]. We think that their extension to interactive multiview 3D video streaming is an important area of future work.

The work in this chapter provides a complementary perspective on the analysis of rendering quality presented in Chapter 2. As demonstrated in Chapter 2, ren-
dering quality depends on the number and selection of textures that are blended together. Our work in this chapter performs additional analysis for the case that the multiple streams are required to address the inherent limitations of real-world system realizations, such as the delay in interactive scenarios. In addition, the proposed algorithm can be readily employed to reduce the interaction latency in streaming systems architected according to the layered on-demand transmission model proposed in Chapter 3. In a system prototype (as in Chapter 3), the feedback channel can be implemented using a persistent transport-protocol connection, thus eliminating the connection-establishment delay when requesting viewpoint changes. Further, an adaptation of the proposed user-interaction model will be necessary for systems such as the one presented in Chapter 3 that use a different interaction interface (e.g., a natural 3D-interface as in [73]). Finally, in Chapter 6, we focus on a related problem of efficiently supporting viewing trajectories that are not limited to a horizontal line. In particular, we show that such interaction scenarios cannot be efficiently supported by using standard communication system architectures. To this end, in Chapter 6, we propose an optimized communication system architecture for 3D video streaming services.
Chapter 6

3D Video Streaming with Remote Rendering

This chapter proposes an architecture for the delivery of multiview 3D video streams to a large number of concurrent users. The proposed architecture is a streaming Content Delivery Network (streaming-CDN) that provides the following services: rendering of virtual views, real-time encoding and streaming. The main insight of our proposal is that the conventional wisdom of regarding a streaming system as a distributed application with distributed data and centralized computation may not be an appropriate model for future multiview 3D video streaming systems. We argue that the alternative view of a multiview 3D video streaming system as an application with distributed data and distributed computation is a better model for cost-effective realizations of a large system with resource-constrained and heterogeneous users. Specifically, our hypothesis in this chapter is that offloading the view-rendering computation to a remote location and providing it as a service of the existing streaming-CDNs is both technologically possible and useful for cost optimizations in 3D video streaming. To support this statement and to validate the hypothesis, our main contribution consists of: (1) analysis of the usefulness of the proposed architecture in the context of resource costs of today’s streaming-CDNs and (2) implementation of a small-scale 3D video streaming prototype according to the remote-rendering architecture. To the best of our knowledge, this is the first multiview 3D video streaming proposal to consider such a distributed system architecture, where we have found that it reduces network bandwidth and that the implementation proved to be technologically feasible.
6.1 Large-scale 3D video delivery-challenge

In this chapter, we argue that the potential of a 3D video streaming system to grow to an Internet-scale service will be conditioned on its ability to cost-effectively use the resources at the server, in the network and at the endpoints. A 3D video streaming service that resolves this challenge will be able to overcome economic barriers to its growth, pertaining to the high cost of resource provisioning. Correspondingly, it will be able to scale up the amount of available resources as the number of users grows. This thesis has already partly addressed the challenge of cost-effective resource use. Specifically, both the layered streaming model proposed in Chapter 3 and the adaptive streaming algorithms presented in Chapter 4 and Chapter 5 focus on the efficient use of network bandwidth, based on our assumption that the network bandwidth will continue to be scarce and heterogeneous in the future Internet. The work in this chapter is complementary to the work presented so far in its focus on a holistic use of heterogeneous resources available in a large-scale 3D video streaming system. We regard all the individual resources in such a system – the available server and network bandwidth, computation power at servers and endpoints – as belonging to a joint pool of resources that can be combined as appropriate. This assumption is reasonable because the distribution of coded multiview data over the network, especially at the level of servers providing content for users is typically operated by the same service provider.

The problem studied in this chapter is twofold:

1. How to significantly reduce the bandwidth requirement for multiview 3D video streaming and support even bandwidth-impoverished receivers in the system.

2. How to minimize the bandwidth cost in a large multiview 3D video streaming system.

To this end, our contribution is as follows:

- We propose a novel multiview 3D video streaming architecture that allows to significantly reduce the bandwidth requirement of a 3D video streaming service and thus the total system cost.

- We validate this architecture with a small-scale streaming prototype.
To show the usefulness of the proposed architecture for a large-scale streaming system, we analyze bandwidth-provisioning costs in such a system as well as technology trends that suggest a large-scale system implementation may be feasible in the near future.

This chapter is structured as follows. Section 6.2 surveys current solutions for large-scale deployment of Internet streaming services and presents an analysis of their cost structure. Section 6.3 motivates a fresh view on multiview 3D video delivery architectures and cost optimizations. In Section 6.4, we detail on the proposed architecture and its useful properties. In Section 6.5, we demonstrate the technical feasibility of 3D video streaming and rendering algorithms within this architecture. We conclude in Section 6.6.

6.2 3D video in conventional delivery architectures

To better understand the problem statement as stated in the previous section, in this section, we first review current solutions for large-scale deployment of Internet streaming services and then present an analysis of their cost structure. In the absence of real-world deployment of a large-scale multiview 3D video streaming service, our analysis is based on a model of a hypothetical service architected according to today’s best practices for large-scale Internet streaming services.

6.2.1 State-of-the-art streaming-CDNs

The state-of-the-art streaming service architecture in today’s Internet is a Streaming Content Delivery Network (streaming-CDN) [28, 29]. A streaming-CDN is a distributed system of servers that provide efficient video caching, replication and streaming services. The individual servers in a streaming-CDN are inter-connected via local or wide-area networks, thus forming an overlay streaming network [135]. Being privately managed, streaming-CDNs provide control over the content stored at individual servers as well as the traffic between the servers. The benefits are direct for both the content providers and end users. For providers, streaming-CDNs provide significant bandwidth savings. Instead of serving one stream per receiver, a single stream needs to be sent to a streaming-CDN, redirecting every

\[\text{As stated in Chapter 1, we take the term "3D video" to mean both the stereoscopic and the multiview video. Although large-scale stereoscopic-video systems exist, multiview services have seen no deployment up to this date.} \]
incoming receiver to one of its servers. The requested videos are streamed to receivers from a selected CDN server, requiring only a unicast network architecture. For end users, the use of streaming-CDNs may lead to a better quality of experience, if the CDN server is selected according to the network metrics such as RTT, topological proximity, or congestion level [28, 163]. Due to these benefits, streaming-CDNs are widely deployed in the Internet, with their aggregate bandwidth capacity sufficient to support millions of users [30].

The main enabling factors for the wide deployment of streaming-CDNs are the decreasing costs of hardware, wholesale bandwidth [186] and the consolidation trend of streaming services. The prices of hardware show a steady downward trend, which enables the operators of streaming-CDNs to build large-scale distributed server systems. Likewise, the decreasing bandwidth prices for large enterprise customers made it possible to admit large traffic between their distributed server locations [186]. The ability to serve the video streams on behalf of a large number of content providers over the same CDN, enables the operators to benefit from a yet lower cost per unit of hardware and bandwidth. As a result, the streaming-CDNs benefit from economies-of-scale and lead to a consolidation of streaming services in that the multitude of Internet video today is served from a small number of large CDNs [26], such as [28] and [29].

Cost-effectiveness of streaming-CDNs. To conclude, today’s wide use of streaming-CDNs owes to steadily decreasing costs of hardware and bandwidth caused by technological advances, as well as the economies of scale associated with the consolidation of streaming services. However, the physical resources in a large streaming-CDN often need to be acquired and provisioned ahead of the demand. Due to a generally unpredictable demand, it is difficult to optimize the cost of resource provisioning. As a result, the key to the cost-effectiveness of streaming-CDNs is the ability to match the costs of resource provisioning to the aggregate bandwidth demand, such that the CDN can continuously operate at high utilization levels [186]. In the case of future 3D video streaming deployments, achieving the cost-effectiveness will be particularly challenging. As our subsequent analysis shows, the bandwidth demand in 3D video streaming is large and leads to a multiplicative bandwidth-cost increase compared to conventional 2D streaming. In addition, this demand is difficult to predict due to its dependence on viewing scenarios and user-navigation patterns. In this chapter, we propose a streaming architecture that is useful in resolving these challenges and allows for cost-effective
system realizations.

6.2.2 Analysis of bandwidth cost for providing a 3D video streaming service

With streaming-CDN operators bearing the bandwidth cost of their distributed server infrastructure, we argue that the feasibility of high-bandwidth 3D video streaming over state-of-the-art streaming-CDNs will depend on the ability to cost-effectively utilize this infrastructure. To better understand the underlying cost, we propose a simple analytical model of the total bandwidth cost of a 3D video streaming service deployed over a streaming-CDN.

Our model computes the total bandwidth cost of a streaming service by multiplying the average server-bandwidth cost per user by a number of concurrent users. To accurately model the bandwidth cost per user, our model takes into account a number of important parameters. These parameters can roughly be divided in three groups. In the first group are the streaming-related parameters that model the bandwidth cost of a streaming server as well as an operational bandwidth overhead, to be defined later, that is required when implementing a streaming service over a specific transport protocol. In the second group, we include parameters that model the impact of a specific bandwidth-provisioning policy in the access network that connects a streaming-CDN to end users. As the access network is outside the administrative domain of a streaming-CDN provider (it is independently administered and provisioned by an Internet-service provider), the impact of a specific provisioning policy needs to be factored in when computing the total bandwidth cost in a CDN. Finally, in the third group we include parameters that are specific to 3D video streaming. These parameters model the bandwidth requirement of a 3D video streaming service, by including the specifics of the viewing scenarios as well as the efficiency of the employed rendering and coding algorithms.

Our model parameters and their typical values are explained in detail as follows.

- **Bandwidth of a single CDN site – \( B_{\text{out}} \):** Single-server unicast system with a server outbound bandwidth \( B_{\text{out}} \), serving 3D videos coded at a rate \( R \) can support \( B_{\text{out}}/R \) concurrent users, assuming an equal rate \( R \) for all videos. While a single streaming server can have an outbound bandwidth \( B_{\text{out}} \) of 1.5 Gb/s [187], a CDN site similar to that employed in today’s VoD IPTV
systems may have an aggregate outbound bandwidth of 150 Gb/s [188]. We assume that a streaming-CDN has multiple geographically-distributed sites [28], totaling $n_s$ servers.

- **Coding rate** – $R$: 3D video representation consists of a number of texture and geometry streams, as in the proposed layered streaming model in Chapter 3. For ease of analysis, the rate $R$ of the coded 3D video representation can be expressed as a multiple of the rate $R_t$ of a single texture layer. For illustration, we provide several sample values for the rate $R$, based on viewing scenarios discussed earlier in this thesis. Stereoscopic video with independent layer coding (Chapter 3) has a rate of $R = 2 \cdot R_t$. Free-viewpoint video with continuous horizontal viewpoint change (Chapter 4) has a rate of up to $R = 4 \cdot R_t$. Free-viewpoint video captured with a 2D camera array and supporting a zoom-out functionality [10, 4, 9] requires a significantly larger bandwidth, e.g., $R = 16 \cdot R_t$ or $R = 32 \cdot R_t$. Similar large values are required in scenarios where multiple virtual views need to be rendered simultaneously for multiview autostereoscopic displays [11]. This also holds for very complex scenes where it may be necessary to dynamically increase the scene-sampling rate [10].

- **Compression-efficiency scaling factor** – $f_{ce}$: The scaling factor $f_{ce}$ accounts for differences in compression efficiency among the coding standards (e.g., MPEG-2 versus H.264/MPEG-4 AVC). This factor may also be used to reflect a higher compression efficiency of coding standards that employ interlayer predictions (e.g., MVC), compared to independent layer coding [21].

- **Streaming-rate scaling factor** – $b_{TCP}$: TCP is the most widely-used transport protocol for video delivery in today’s streaming-CDNs [26, 36]. As discussed in Chapter 4, a guideline for unicast streaming over TCP is to stream the video at a rate two times lower than TCP-estimated available bandwidth [42]. This means using only 50% of the unicast channel capacity on average, or equivalently, streaming the video at a reduced quality. This conservative streaming rate is recommended in order to avoid receiver-buffer starvations due to temporary bandwidth drops and provides a statistical guarantee for uninterrupted streaming playback [42]. In contrast, the adaptive streaming techniques presented in Chapter 4 would temporarily switch to a lower rate when the bandwidth drops and switch back to the higher rate
when the available bandwidth recovers. For this analysis, we assume a conservative rate of $b_{TCP} = 2$ at all times during transmission. This accounts for the most common case and gives us an upper bound for the streaming overhead.

- **Oversubscription ratio – $b_d$:** Access networks are often over-subscribed [34]. The over-subscription is a bandwidth-provisioning practice where the capacity of an access network is smaller than the sum of capacities of individual access links. This practice allows the network providers to reduce cost and is common in today’s Internet [34]. As a result, the available bandwidth for a single user of a 3D video service may be lower than nominal during peak traffic hours, due to the competing traffic from other service users. A typical value of bandwidth reduction during peak hours is 25% [34], i.e., $b_d = 1.25^2$.

- **Bandwidth cost – $c$:** Operational streaming-CDN cost consists of bandwidth costs ($c_b$) and maintenance costs ($c_m$), thus $c = c_b + c_m$. The cost of bandwidth for large enterprise customers can be estimated from industry-wide surveys [190]. We assume that the bandwidth costs and maintenance costs equally contribute to the total cost, i.e., $c_b = c_m$.

Having discussed the important parameters, we derive our analytical model as follows. The average number of concurrent users in a streaming-CDN is primarily determined by the sum of bandwidths available at each CDN site, which is given as $n_s \cdot B_{out}$. However, for an accurate estimation, the model needs to factor in the bandwidth efficiency of the actual codec and the transport protocol used in the system. Employing an inefficient codec or a transport protocol that requires a large bandwidth margin against receiver-buffer starvation, effectively reduces the number of users that the system can support. We model this reduction as a product of scaling factors $f_{ce}$ and $b_{TCP}$. In turn, due to over-subscription in access networks, some users are not able to fully utilize their access-bandwidth capacity, which effectively increases the number of users that a streaming-CDN can support. We model this increase as a scaling factor $b_d$. Having included these scaling factors, we compute the average number of users by dividing the scaled sum of bandwidths by the rate of an average user $R$. Thus, the average number of

---

2This is the value observed in today’s access-network deployments and should not be confused with the maximum allowed over-subscription ratio [189].
3D video streaming users that can be concurrently supported in a streaming-CDN can be approximated as:

\[ n_u = \frac{n_s \cdot B_{out}}{R \cdot f_{ce} \cdot b_{TCP} \cdot b_d}. \]  

(6.1)

Correspondingly, the total bandwidth cost of supporting \( n_u \) concurrent users can be approximated as follows:

\[ C = n_s \cdot B_{out} \cdot c = n_u \cdot R \cdot f_{ce} \cdot b_{TCP} \cdot b_d \cdot c. \]  

(6.2)

Table 6.1 provides an overview of the parameters used in Eq. (6.1) and Eq. (6.2).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_u )</td>
<td>Number of concurrent users</td>
</tr>
<tr>
<td>( n_s )</td>
<td>Number of CDN sites in a streaming-CDN</td>
</tr>
<tr>
<td>( B_{out} )</td>
<td>Outbound server bandwidth (single CDN site)</td>
</tr>
<tr>
<td>( R )</td>
<td>3D video coding rate</td>
</tr>
<tr>
<td>( f_{ce} )</td>
<td>Compression-efficiency scaling factor</td>
</tr>
<tr>
<td>( b_{TCP} )</td>
<td>Streaming-rate scaling factor</td>
</tr>
<tr>
<td>( b_d )</td>
<td>Network-bandwidth oversubscription ratio</td>
</tr>
<tr>
<td>( c )</td>
<td>Operational bandwidth cost per Mb/s</td>
</tr>
<tr>
<td>( C )</td>
<td>Total bandwidth cost</td>
</tr>
</tbody>
</table>

Table 6.1: Parameters for the computation of bandwidth cost for 3D video streaming.

Equation (6.2) shows that the total bandwidth cost of a streaming-CDN is linearly dependent on the coding rate \( R \) of the employed 3D video representation. The key challenge here is that a 3D video representation consists of multiple texture and geometry streams. As a result, the rate \( R \) in a multiview 3D video system is a multiple of the coding rate of a single layer \( R_t \). To put this in a perspective, our results in Chapter 4 and Chapter 5 suggest that the coding rate of a single layer may be as high as 8 Mb/s for sufficient display quality, employing state-of-the-art compression and virtual-view rendering algorithms. In comparison, today’s top-of-the-line streaming-video offerings in the Internet are encoded at rates of 4–6 Mb/s [36, 37].

We regard this multiplicative cost increase in multiview 3D video streaming as the major challenge to deployment of large-scale 3D video streaming in the future.
Internet. This challenge is evident today in view of the high costs of providing bandwidth as a network service. Importantly, in our view, this challenge will remain in the years to come, even in a very favorable scenario where the wholesale streaming-CDN bandwidth costs continue to decrease at the same rate as in the recent years (tenfold decrease is observed in the period 2006-2010 [186]).

6.3 3D video streaming as a distributed rendering application

We propose to address the multiplicative increase of bandwidth cost with a new system architecture. This proposal is based on an observation that the existing streaming architectures fundamentally limit our ability to reduce this cost. This limitation and the proposed solution are discussed next.

The state-of-the-art compression standard for multiview video – MVC – is more efficient for certain multiview 3D video sequences than an independent coding of views using H.264/MPEG-4 AVC [21]. As this thesis shows, significant further gains are possible by using efficient 3D data-representations and rendering algorithms (Chapter 2), layered scene representations and streaming models (Chapter 3) and efficient streaming algorithms (Chapters 4 and 5). Despite the efficiency of these compression and streaming solutions, the size of a compressed layered 3D video representation is still linearly proportional to the number of layers.

Common to our contributions and related literature solutions is a view of a streaming system as an application with distributed data and centralized computation. In this view, a streaming system merely transports the data from the capturing endpoint to the display endpoint, which executes the computations for decoding and rendering. Correspondingly, the contributions and related literature assume that the bandwidth resource is scarce and propose algorithms to use it efficiently. Most importantly, these solutions manage the multiplicative cost increase by limiting the set of possible viewing scenarios. As discussed in Section 6.2.2, viewing scenarios exist where the view rendering needs to support user-navigation trajectories beyond a horizontal line and even generate multiples of such virtual views simultaneously. Solutions that do not address the full complexity of the multiplicative cost-increase should be considered local system optimizations. The solution proposed in this chapter is an architecture that allows global system
optimizations to be implemented.

Central to our proposal is a view of the 3D video streaming system as an application with distributed data and distributed computation. In particular, we show that different architectural options exist for implementing view rendering, the distinct functionality of 3D video streaming systems. In doing this, we borrow concepts from distributed computer-graphics research.

Distributed graphics applications often render complex synthetic scenes by partitioning the rendering load between the server and the receivers [191]. The overview in [191] identifies three broad architectural approaches for partitioning the rendering load. We extend the overview in [191] with examples of recent systems implemented according to each architecture. For simplicity, we consider a scene description consisting of four streams, as depicted in Figure 6.1.

1. **Local rendering.** A rendering architecture where a server transmits all available scene-description data to the receiver for rendering is shown in Figure 6.1 (a). This architecture is analogous to that of today’s video streaming systems.

2. **Remote rendering.** This architecture is common in computer graphics for implementing complex rendering tasks (Fig. 6.1 (b)). The architecture is particularly useful if a receiver does not have the resources to render the full scene description, while rendering a simplified description is unacceptable. In this case, a powerful remote server renders the model and sends the resulting high-quality images to the receiver [192]. For bandwidth efficiency, the rendered images are compressed at a sufficient quality.

3. **Hybrid rendering.** As a middle ground between the above two architectures, the hybrid methods balance the rendering load between the receiver and the server (Fig. 6.1 (c)). The server renders a part of the scene description, while the receiver combines those renderings with the locally-rendered scene parts [193, 111].

When applied to a multiview 3D video service, these architectures lead to significantly different bandwidth costs. For example, in a remote-rendering architecture, the bandwidth cost at the receiver is significantly reduced. The cost then only includes receiving and decoding a single stream of rendered images. Likewise, the server-bandwidth cost reduces to transmitting a single stream. For a more formal analysis, we can directly compare the total bandwidth costs of a 3D video
Figure 6.1: Models for distributed 3D video delivery.
service for the three architectures. Without a loss of generality, our comparison considers the following scenario. All four views depicted in Fig. 6.1 are virtual, as a particularly challenging case, and positioned along an arc connecting the camera centers in a multi-camera capturing system. Scene navigation is one-dimensional in the horizontal direction, where a user requests all four views at once. The 3D data-representation of the scene consists of texture and disparity layers and the rendering algorithm from Section 2.2.4 is used to synthesize the views. We assume equal coding rates for the texture and disparity layers and for simplicity, ignore the overhead of occlusion-layer coding. The total coding rate per virtual view can then be approximated as \(3 \cdot R_t\), where \(R_t\) is the rate of a single texture layer. Using Eq. (6.2) with parameters as defined in Section 6.2.2, we can compute the total bandwidth costs for the three architectures as follows.

1. The *local-rendering* architecture directly corresponds to the conventional delivery model used in today’s video streaming systems. The total bandwidth cost can be directly estimated from Eq. (6.2) as:

   \[
   C_{\text{local}} = n_u \cdot 12 \cdot R_t \cdot f_{\text{ce}} \cdot b_{\text{TCP}} \cdot b_d \cdot c. \tag{6.3}
   \]

2. In the *remote-rendering* architecture, all four requested views are synthesized at the server end. Correspondingly, the bandwidth cost is:

   \[
   C_{\text{remote}} = n_u \cdot 4 \cdot R_t \cdot f_{\text{ce}} \cdot b_{\text{TCP}} \cdot b_d \cdot c. \tag{6.4}
   \]

3. For the *hybrid* architecture, we compute a sum of the coding rates of the views rendered remotely and those rendered locally. In our example in Fig. 6.1 (c), we render the two inner views remotely and the outer two views locally. Therefore, the total cost is:

   \[
   C_{\text{hybrid}} = n_u \cdot 8 \cdot R_t \cdot f_{\text{ce}} \cdot b_{\text{TCP}} \cdot b_d \cdot c. \tag{6.5}
   \]

This example illustrates the bandwidth-cost savings achievable with remote and hybrid architectures, as compared to the conventional local-rendering architecture. Clearly, our example is a single point in the space of possible application scenarios. As discussed in Section 6.2.2, the achievable savings are application-dependent. All other factors in Table 6.1 being equal, the bandwidth-cost savings critically depend on the viewing scenario (e.g., dimensionality of the navigation
trajectory) as well as on the efficiency of the employed coding and rendering algorithms. Our experience suggests that the impact of both factors can be conveniently approximated as a linear dependence on the coding rate of a single texture layer. However, we yet need to develop a strong theoretical basis to underpin this experimental finding.

The trade-off in the remote- and hybrid-rendering architectures is that the bulk of the computation is offloaded to the server end. The computation cost at the server includes the costs of rendering the scene description and encoding the rendered images in real-time for streaming. Therefore, the choice among different architectures illustrates a trade-off between the computation load at the server (and the receiver) and the network bandwidth. Distributed graphics applications manage this trade-off well [191]. The key to their effectiveness is that the rendering computation is distributed – both the server and the receiver have the rendering capability such that a part of the rendering logic can execute at the server and another part at the receiver. This gives the system an additional freedom to execute its computation where most appropriate and optimize its use of distributed resources. In the sequel, we detail on how such a distributed architecture can be adapted for large-scale multiview 3D video delivery.

6.4 Proposed system architecture

Our contribution in this chapter is a 3D video streaming architecture that supports both the remote-rendering model (Fig. 6.1 (b)) and the state-of-the-art local-rendering model (Fig. 6.1 (a)). Therefore, the novelty of our contribution is in the extension of a state-of-the-art delivery architecture with the remote-rendering model in Fig. 6.1 (b). For completeness, we note that the hybrid-rendering model (Fig. 6.1 (c)) is most comprehensive and the most likely candidate for future deployment of 3D video streaming services in the Internet. However, as the remote-rendering functionality is the distinct new architectural feature of both the hybrid and the remote model as compared to the state-of-the-art local model, we focus on this functionality first and implement it for the simpler of the two models, i.e., the remote-rendering architecture. We visualize this in Figure 6.2 in order to help the reader quickly place each discussed delivery model in the context of our original contribution in this chapter. A detailed presentation of the proposed 3D video streaming architecture is provided next.
6. 3D Video Streaming with Remote Rendering

6.4.1 3D video streaming with remote rendering

Our proposed architecture adopts the key insight from distributed graphics applications – that a distribution of computation logic between the server and the receiver allows to balance both the network bandwidth and the computation load. We apply this insight to propose an extension of state-of-the-art streaming-CDNs as follows. First, we regard the individual resources in a streaming-CDN – the available server and network bandwidth, computation power at servers and receivers – as belonging to a joint pool of resources that can be combined as appropriate. We then propose to extend a streaming-CDN with processing modules to enable the remote-rendering functionality (Fig. 6.1 (b)). These processing modules implement view rendering and real-time encoding of rendered views. In this way, a 3D video streaming system becomes a distributed application with both the data and the computation distributed between the servers of a streaming-CDN and the receivers. The benefit is the additional freedom to execute the computation where most appropriate in the end-to-end system and optimize the use of distributed resources. As a result, the proposed architecture allows to globally address the multiplicative-cost challenge inherent in 3D video streaming. We note that besides the bandwidth cost (Eq. (6.2)), this may also include the multiplicative cost of processing for virtual-view rendering at the receiver in traditional local architectures.

The proposed architecture is illustrated in Figure 6.3. It consists of a 3D video server, a 3D streaming-CDN and the receivers. The distinct new capabilities of a 3D video server and a 3D streaming-CDN, as compared to server and CDN components in conventional 2D video systems, are as follows.

- **3D video server.** The 3D server is a part of the content provider’s network, located at the point of connection to a 3D streaming-CDN. For each re-

![Figure 6.2: 3D video streaming architectures.](image-url)
quested 3D video, it generates the corresponding 3D video representation consisting of multiple texture and geometry streams, and injects it into a 3D streaming-CDN.

- **3D streaming-CDN.** The 3D streaming-CDN is a distributed network of servers equipped with processing modules required to implement the remote-rendering delivery model. The key processing module in this context is *view rendering* that synthesizes virtual views corresponding to each receiver’s viewing parameters. The functionality of the view-rendering module is supported with real-time decoding and encoding modules. The decoding module decompresses the texture and geometry streams required for view rendering. The encoding module compresses the synthesized view before streaming it to the requesting receiver. In addition, similarly to conventional 2D video systems, each server in a 3D streaming-CDN can directly stream the 3D video representation to requesting receivers. In this way, the architecture also supports the traditional local delivery model.

In the next section, the most important system aspects are discussed in detail. For convenience, our discussion refers to a live 3D video streaming scenario. Besides this, a VoD 3D video streaming can be supported using the same architecture.

### 6.4.2 3D server

The back-end functionality of a 3D server is decomposed into 3D scene recording, scene modeling and compression stages. A 3D scene is recorded with an array of static, fully calibrated cameras. The raw camera frames and their calibration data are first processed at the scene-modeling stage, to extract a geometric description of the scene. For example, the extracted geometry streams may provide enough

---

**Figure 6.3:** Proposed 3D video streaming architecture.
local information for the view-rendering algorithm to support a continuous horizontal navigation, as in Chapter 2. The resulting 3D video representation consists of a number of texture and geometry streams. As in Chapter 3, we assume a layered streaming model where each layer carries a single coded video signal or coded scene-description data.

6.4.3 3D streaming-CDN

We describe the functionality of a 3D streaming-CDN using an example illustrated in Figure 6.4. In this example, we consider a small session with three receivers concurrently accessing the same multiview 3D video. All three receivers are concurrently expressing their interest in navigating a region of the 3D scene spatially delimited by Cameras 1 and 3.

Each receiver joins the session by sending a request to the 3D server. The server replies with a list of available camera viewpoints, their calibration parameters and geometry streams. Each receiver is then redirected to a selected streaming edge server (receiver A and receiver B to the Node D, whereas receiver C is redirected to Node E). The streaming edge servers act as agents for their receivers, fetching the streams as needed to support receiver interaction with the scene. A proper selection of the streaming edge server results in a low latency for the receiver [170, 163]. Due to continuous interactivity, the server latency is an important parameter for the overall user’s quality of experience (Chapter 5).

The 3D streaming-CDN connects to a 3D server via one of its edge servers (Node A in Figure 6.4). The 3D server uses the connection to Node A to inject the texture and geometry streams describing the scene region between Cameras 1 and 3. These streams propagate through the 3D streaming-CDN as follows. Receivers A and B are navigating the scene region between Camera 1 and Camera 2, while receiver C is at a position between Camera 2 and Camera 3. The 3D streaming-CDN propagates the required streams to the selected edge servers. Note that each stream propagates only once, while the 3D streaming-CDN efficiently replicates the streams at intermediate servers. This way, the bandwidth cost ideally depends on the number of active viewpoints, but is independent of the number of receivers. Effectively, the streaming-CDN implements a multicast delivery among its servers, thus significantly reducing the bandwidth cost. Further, the stream propagation through a privately-managed streaming-CDN occurs often with a lower delay and higher reliability than in the public Internet [28].
Figure 6.4: 3D streaming-CDN: a distributed 3D video delivery architecture that combines the local and the remote-rendering model.
Each edge server is equipped with processing modules for local rendering (streaming, caching) and extended with modules required for remote rendering (decoding, view rendering, encoding). In our example, these two delivery models are supported as follows.

- **Remote rendering.** In Figure 6.4, Node D performs the view rendering for receivers A and B. In this example, Node D first fetches the texture and geometry streams required to render virtual views in the scene region delimited by Camera 1 and Camera 2. The fetched streams are decoded, followed by a rendering of two different virtual views, corresponding to viewing parameters of receivers A and B. Next, for each view, the rendered frames are encoded and transmitted to the requesting receiver as a single video stream.

- **Local rendering.** In Figure 6.4, our 3D streaming-CDN implements the local rendering model for receiver C. Correspondingly, the texture and geometry streams required for rendering a view in the scene region between Cameras 2 and 3 are fetched by the edge server Node E on behalf of receiver C. Although Node E is equipped with the required processing modules, it does not perform view rendering. Instead, Node E directly forwards the fetched texture and geometry streams to receiver C, which decodes them and renders the requested view locally.

In this example, it is important to note that the distributed architecture of a 3D streaming-CDN allows to implement the remote-rendering delivery model in a scalable fashion. Each edge server only handles a fraction of the total viewer population in the system.

### 6.4.4 Usefulness of a 3D streaming-CDN

The proposed architecture is useful for a cost-effective large-scale 3D video delivery due to its following properties.

- Rendering of virtual views provided as a service of the 3D streaming-CDN allows to reduce the bandwidth and computation costs. The reduction is from linear in the number of required layers potentially to a constant. In comparison with traditional streaming architectures, this reduction is achieved for the outbound bandwidth of streaming edge servers, the incoming bandwidth of receivers and the computation at receivers. Importantly, the total
required bandwidth in a remote-rendering architecture is independent of the viewing scenarios and user-navigation patterns. As a result, the proposed architecture allows to manage the multiplicative cost-increase. This architecture also allows for a flexible resource control. For example, the system can provide rendering services for a certain fraction of receivers. Additionally, it can adjust its bandwidth cost dynamically, by switching between remote and local rendering for different receivers or for the same receiver over time.

- The trade-off in the proposed architecture is an increased computation cost on edge servers of a 3D streaming-CDN in comparison with traditional architectures. This includes the costs of decoding, rendering and encoding at streaming edge servers. The encoding and rendering costs are linear in the number of users, while the decoding cost is proportional to the number of layers. In absolute terms, the dominant among them is the encoding cost [194], followed by decoding and rendering costs [12]. Although the resulting cost may be significant, we believe that the trade-off is reasonable and that benefits will outweigh the cost in many practical scenarios. In particular, the total system cost is borne by the streaming-CDN provider, including the bandwidth and computation costs. A 3D streaming-CDN provider can benefit from economies-of-scale, optimize its bandwidth-computation trade-off and expose it through service charges [195]. From the technical-feasibility standpoint, our small-scale prototype in Section 6.5 demonstrates a real-time end-to-end system performance. Further, in Section 6.5.4, we provide an overview of the technology trends that allow for efficient large-scale implementations of the proposed architecture.

Additional properties of the proposed architecture that are useful for performance enhancements of 3D video streaming systems are as follows:

- Bandwidth among the streaming-CDN servers is used efficiently due to application-layer multicast inside the CDN. Our architecture inherits this property from the state-of-the-art streaming-CDNs. The use of our layered streaming model (Chapter 3) ensures that only the requested layers are active in a streaming-CDN, thus further increasing the bandwidth efficiency.

- Geographical distribution of streaming edge servers in a CDN allows for a latency-optimized server selection, which reduces the average response
latency in the system and improves the user’s sense of immersion.

• The proposed architecture is beneficial for addressing the challenge of receiver heterogeneity at the granularity of a streaming layer (Chapter 3). For different receivers, the system can balance the number of layers streamed directly with remote rendering.

Summarizing, the proposed architecture allows to flexibly manage the multiplicative cost-increase of bandwidth in 3D video streaming. This includes an implementation option to reduce the bandwidth cost to a constant in the number of views, thus equaling the cost of conventional 2D video streaming. In addition, the proposed architecture can be used as a basis for implementing global performance enhancements in a system, most notably with respect to interaction latency and time-varying bandwidth availability. The trade-off in the proposed architecture is an increased computation cost for a streaming-CDN, caused by implementing view rendering and encoding as CDN services. As this cost is borne by the streaming-CDN provider, the proposed architecture allows to optimize the trade-off between computation and streaming bandwidth at the system level. In this way, globally-optimal bandwidth allocations can be found, thereby optimizing the total system cost.

6.5 Prototype implementation of remote view rendering and streaming

To demonstrate the feasibility of the proposed architecture, we implement a small-scale prototype of 3D video streaming using the remote-rendering functionality. The prototype implements monoscopic multiple-perspective viewing of a remote 3D scene and user interaction. It demonstrates the proposed architectural view of a streaming system as a distributed application in which the server creates the rendered views, compresses and streams them to the receiver in real time.

Our prototype implementation relies on algorithms and methods developed earlier in this thesis – the layered scene representation from Chapter 3 and the efficient 3D representations, compression and rendering algorithms from Chapter 2. In particular, we reuse software components developed for the prototype in Chapter 3 and adapt them for use in a remote-rendering architecture. In the sequel, we describe the components for rendering, encoding, streaming and user interaction, as well as their integration in the system depicted in Fig. 6.5.
6.5.1 Remote rendering, coding and streaming at the server end

The server in the proposed 3D streaming-CDN implements components for decoding, virtual-view rendering, encoding and streaming (Fig. 6.5). Our CDN-server implementation assumes that all scene-description data required for rendering are locally available at the server. This includes textures, geometry layers (depth or disparity maps) and optional scene-description layers. We note that a local availability may not be given in all real-world scenarios, such that the server may have to fetch these data from another CDN server and decode them before rendering (Fig. 6.3). In our case, the uncompressed data are available and stored in server memory.

**Virtual-view rendering** is performed using the algorithm from Section 2.2.4, implemented as in Chapter 3. The employed rendering configuration enables a continuous horizontal navigation between pairs of original cameras. For each virtual viewpoint, the input to the algorithm includes: (1) the left texture stream, (2) the right texture stream, (3) per-pixel disparity, and (4) occlusion layer (pixels flagged as visible to one of the cameras only). The algorithm synthesizes virtual frames at an arbitrary horizontal position between the original cameras.

**Encoding.** Each synthesized virtual frame is read out from the GPU’s framebuffer, using the `glReadPixels()` method of the OpenGL API [122]. This step is accompanied with a color-space conversion (RGB to YUV) for every frame in order to prepare the frames for encoding. The frames are passed in sequence to the encoding component through a named pipe. We use an MPEG-4 codec accessible through the Advanced Systems Format (ASF) container of the ffmpeg libraries [196]. This codec is fully compliant with the Simple Profile (SP) of the MPEG-4 standard [16]. We employ the codec’s default rate control to

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Texture+Disparity</th>
<th>Texture+Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Objects”</td>
<td>“Ballet”</td>
<td></td>
</tr>
<tr>
<td>Frame rate</td>
<td>20 fps</td>
<td>25 fps</td>
</tr>
<tr>
<td>Resolution</td>
<td>320×240 pixels</td>
<td>1024×768 pixels</td>
</tr>
<tr>
<td>Video codec</td>
<td>MPEG-4</td>
<td>MPEG-4</td>
</tr>
<tr>
<td>Rate control</td>
<td>CBR</td>
<td>CBR</td>
</tr>
<tr>
<td>Bitrate</td>
<td>1 Mb/s</td>
<td>10 Mb/s</td>
</tr>
</tbody>
</table>

**Table 6.2:** Test sequences and encoding parameters for the remote-rendering prototype.
Figure 6.5: Prototype implementation of multiple-perspective viewing with remote rendering, encoding and streaming of virtual views.
obtain Constant-Bitrate streams (CBR). The main encoding parameters are given in Table 6.2.

Streaming. We use the same protocol stack as in our implementation from Chapter 3. Thus, we use UDP at the transport layer, Real-Time Transport Protocol (RTP) [115] as the transport-layer extension and Real-Time Streaming Protocol (RTSP) [116] at the application layer. We use the live555 [113] software library to implement the stack. Since a single coded video stream is transmitted, no extensions to live555 are required. Thus, we directly reuse the encapsulation, packetization and transmission components of the library.

User interaction. The user navigates the scene by moving the mouse pointer inside of his video display-window. A virtual view is rendered that matches the user’s viewing parameters. Our implementation of the interaction component is as described in Section 3.4.2, with the following difference. For simplicity, our implementation of the feedback channel does not rely on the service-oriented framework as in Section 3.4.5. Instead, our prototype emulates the feedback channel by generating view-switching requests directly at the server end.

6.5.2 Decoding and display at the receiver end

At the receiver end, we implement components for decoding, rendering and display adaptation (Fig. 6.5). As only a monoscopic video stream is received and displayed (containing a sequence of already synthesized virtual frames), the required receiver processing can be performed by any media player that supports the ASF specification. We employ the widely-used vlc video player [121] for this purpose, since it supports depacketization, decoding and display of monoscopic video streams according to ASF.

6.5.3 Results of the small-scale prototype implementation

To validate the feasibility of the proposed streaming with remote view-rendering, we design an experimental scenario that shows the following properties.

- We demonstrate that our implementation and integration of system components results in a 3D video streaming system that achieves a real-time performance end-to-end.

- We demonstrate that due to the remote-rendering functionality, a multi-view 3D-scene navigation is possible on resource-impoverished receivers at
the bandwidth cost equal to that in conventional 2D video (single view) streaming.

We perform streaming tests with two multiview sequences, “Objects” and “Ballet”, as given in Table 6.2. For both sequences, we reconstruct the disparity and occlusion maps using the algorithm in [76]. The disparity and occlusion information are required by our rendering algorithm. A continuous sequence of virtual views is synthesized according to view-switching requests.

The streaming experiments are performed on a LAN in our department. The server components are deployed on a 3-GHz Desktop PC with a state-of-the-art graphics card \(^3\). The receiver is an outdated laptop (single-core 1.5-GHz CPU) that runs \textit{vlc} player for decoding and display.

We achieve real-time end-to-end performance with both sequences. The viewpoint change is effective at the receiver after a delay of 2 s, which is caused by a buffering delay in \textit{vlc} software and is not an artifact of our implementation.

This experiment illustrates that a streaming system implemented according to the proposed remote-rendering architecture achieves a real-time performance even on commodity server-hardware. Further, it shows that multiple-perspective viewing is feasible even on receivers impoverished in bandwidth and rendering resources.

### 6.5.4 Implementation scenarios for a large-scale deployment

The proposed architecture is based on our paper published in 2007 [51] and reflects the then state-of-the-art in large-scale streaming architectures. More recently, we have observed several technology trends that first, provide new and important evidence for the feasibility of the proposed architecture and second, may allow for efficient large-scale implementations in the near future. We provide an overview of these technology trends in this section.

- **Cloud computing and a convergence of streaming-CDNs and “Clouds”**. Today we are witnessing a significant growth of large-scale distributed computing systems, popularly referred to as “Cloud computing” [92]. The Clouds provide remote computation and storage facilities as services on a per-use charging basis. Increasingly, Cloud infrastructures are employed to provide large-scale streaming services [170, 197]. At the same time, existing

\(^3\)Commercially available under Nvidia GeForce 7600GS.
streaming-CDN providers upgrade their infrastructures to offer on-demand computation services [198]. In our view, this convergence of Clouds and streaming-CDNs illustrates an architectural trend towards a single infrastructure that offers both the streaming and the computation services. The proposed 3D streaming-CDN naturally fits in this infrastructure trend, as it provides both the computation (virtual-view rendering and encoding) and the multiview streaming services. In turn, the availability of a single service infrastructure allows to optimize the trade-off between the costs of computation and streaming bandwidth for future large-scale 3D video streaming services implemented according to our proposed architecture.

- **Multimedia processing as a service.** We are also observing an increased deployment of multimedia-processing services that include large-scale offline encoding of video content [199, 200], as well as remote rendering and real-time encoding of gaming content [201]. We regard these services as examples of the technical feasibility of deploying large-scale services with an architecture similar to the one proposed in this chapter. Further, the actual implementations of these multimedia-processing services are a valuable software basis for implementing services that operate reliably and dependably at a large scale. A large-scale 3D video streaming service can be implemented by using the same software infrastructures and extending them with processing components specific to multiview video processing, most notably virtual-view rendering, multiview coding and streaming algorithms.

- **Availability of specialized processors in streaming-CDNs and Clouds.** The Cloud-computing infrastructures are increasingly being constructed with special-purpose hardware processors such as GPUs [202]. While the primary motivation today is leveraging the GPUs for high-performance computing [203], the availability of GPUs in streaming-CDNs allows for efficient large-scale implementations of rendering and encoding in our proposed architecture. Specifically, the availability of GPUs in streaming-CDNs and Cloud-computing infrastructures is relevant to us for three reasons. First, view-rendering performance can be greatly improved by offloading specific steps of the rendering algorithm onto GPU (e.g., coordinate-system transformations and blending). Second, GPU acceleration for encoding the synthesized views may be necessary to attain real-time performance at high video resolutions. Third, software infrastructures developed to enable a con-
current use of a GPU by multiple services in virtualized Cloud-computing environments can be directly reused to implement a multi-user 3D video service.

- **Parallel computing.** The shift towards parallel computing in distributed [92] and multi-core systems [204] is beneficial for efficient implementations of the proposed architecture. In particular, parallel implementations of the rendering and coding algorithms can significantly improve system performance by exploiting the parallelism that exists at several levels. First, in a large multi-user system, task parallelism exists at the user-level, where the processing for each user can proceed independently from other users. Second, parallelism in a remote-rendering scenario (Fig. 6.1 (b)) can also be exploited at the view-level, such that different views are synthesized and encoded in parallel. In this case, the processing can be distributed either over multiple cores, or over multiple physical servers using the same virtualized Cloud-computing infrastructure. Third, our experiences with virtual-view rendering in Chapter 2 suggest that parallelism also exists at the frame-level, such that certain steps in a rendering algorithm can be executed for each pixel independently (e.g. blending). Finally, the benefits of a parallel implementation are not limited to the remote-rendering architecture in Fig. 6.1 (b). The parallelism at the view- and frame-level can also be exploited for efficient multi-stream decoding and rendering in the traditional local-rendering scenario (Fig. 6.1 (a)).

- **Potential deployment of IP-multicast and high-quality peer-to-peer streaming services.** The ongoing deployment of the IPv6 protocol [24] may lead to a wider use of IP-multicast network architectures. This will be beneficial for (1) addressing the multiplicative increase of bandwidth cost at the network layer and (2) an efficient routing layer in our proposed architecture. However, it would require to resolve the issues involved with operational complexity of multicast routing, since the number of multicast groups in 3D video streaming will be large and group changes are frequent due to interactivity. Additionally, in the near future, today’s low-quality peer-to-peer streaming services [205] based on improved designs [206], may become a viable complement to the proposed 3D streaming-CDN for the delivery of high-quality 3D video streaming.
6.6 Conclusions

In this chapter, we have addressed the challenge of multiplicative increase of bandwidth cost in 3D video streaming, as compared to conventional 2D video streaming. In order to grow to an Internet-scale service, 3D video streaming needs to enable unrestricted 3D-scene viewing on a variety of endpoints, including bandwidth-impoverished receivers. In addition, the service needs to support 3D video rendering for a large number of endpoints simultaneously and to do this in a cost-effective manner. However, we argue that the state-of-the-art large-scale streaming infrastructures lack rendering support for implementing cost-effective 3D video streaming services.

To overcome this limitation, we propose a fresh view of 3D video streaming as a distributed system, in which the rendering logic can be implemented either remotely as a service of the delivery network, or locally at the receiver. What follows from this distributed model is a novel streaming architecture, where the edge servers in a streaming-CDN provide rendering and encoding services for the receivers. The proposed architecture allows to reduce the bandwidth cost from linear in the number of required layers potentially to a constant. This reduction is achieved for: (1) the outbound bandwidth of streaming edge servers, (2) the incoming bandwidth of receivers and (3) the computation at receivers. The trade-off is an increased computation cost on edge servers of a 3D streaming-CDN. We believe that this trade-off is reasonable, since the streaming-CDN provider can first, benefit from the decreasing costs of hardware and network bandwidth by consolidating the bandwidth and computation services and second, optimize these costs globally in the system. We argue by analysis that the proposed architecture is useful for a cost-effective large-scale 3D video delivery, due to its properties summarized in Section 6.4.4. This analysis is complemented with an overview of recent technological developments, that provide important evidence for the feasibility of the proposed architecture and allow for efficient large-scale system implementations in the future. More specifically, these developments involve: cloud computing, multimedia processing as a service, availability of specialized processors in streaming-CDNs, parallel computing and potential deployment of IP-multicast and high-quality peer-to-peer streaming systems.

To demonstrate the technical feasibility of 3D video streaming within the proposed architecture, we have built a real-time streaming prototype. We rely on algorithms and methods developed earlier in this thesis – the layered scene
representation and streaming model from Chapter 3 and the efficient 3D representations, compression and rendering algorithms from Chapter 2. The prototype implements 3D video streaming according to the remote-rendering architecture, including sender-based view rendering, encoding and streaming. Our implementation achieves a real-time end-to-end performance and supports interactive scene navigation. Importantly, we show that this result can be achieved by using a commodity PC as a sender and an old laptop as a receiver. In this way, we experimentally show that the proposed architecture allows to implement a real-time 3D video streaming service for resource-/cost-impoverished receivers. More generally, this prototype shows that the key property of the proposed architecture, the ability to remotely render and stream user-selected virtual views in real time, is possible on commodity hardware. As such, the prototype can be used as a reference implementation for cost-effective rendering of virtual views at both the server end and at the endpoints.

In addition to the presented analysis and achieved results, we note that a 3D video streaming system implemented according to the proposed architecture may have further potential performance benefits. These include a lower latency enabled by a suitable edge server selection, and a smaller jitter due to optimized path routing in a streaming-CDN. To fully realize this potential, the adaptive 3D video streaming algorithms proposed earlier in this thesis can be employed with this new architecture. For example, a suitable edge server selection can be combined with user-adaptive prefetching algorithm from Chapter 5 to further reduce the interaction latency, or to control the average interaction latency in the system. Specifically, a 3D streaming-CDN may apply a provisioning policy that trades the degree of consolidation for the degree of geographical distribution [167]. Under such a policy, even if receiver requests are redirected to an edge server with suboptimal latency [170], a good average performance can be maintained by employing adaptive prefetching. Further, the bandwidth-adaptive algorithm from Chapter 4 can be directly used for adaptations at edge servers or inside a 3D streaming-CDN. This algorithm can also be a basis for novel adaptation algorithms that dynamically switch between remote and local rendering in response to time-varying quality of streaming and rendering services provided by a 3D streaming-CDN.

Besides this, we summarize the most important architectural features that support our interest in the approach followed in this chapter. First, the proposed
architecture is flexible in that the virtual views can be rendered at both the server end and at the endpoints. As a result, the service provider can implement global cost or performance optimizations in the system. For example, the system can provide rendering services for a certain fraction of receivers, or adjust its bandwidth cost dynamically, by switching between remote and local rendering for different receivers. Second, the architecture allows to increase the rendering quality of the service. Namely, a local availability of high-fidelity texture and depth streams at a server, combined with the large computation power and specialized processors at the server end, allow to render virtual views at a visual quality that cannot be achieved with local rendering alone.

Since our original proposal in 2007 [51], there has been relatively little related work in the community. A notable exception is the work in [207]. One reason may be the lack of widely available software to implement high-quality 3D video streaming systems, a challenge we overcome as described in Chapter 3. Another reason may be that large-scale streaming and computing systems are a recent development and that “Clouds” and streaming-CDNs are yet to capture the interest of signal-processing researchers [208]. The system infrastructures to make our vision a reality are being broadly deployed today. We therefore believe that an important area of future research is to actually build a complete 3D video delivery system according to the proposed architecture.
6. 3D Video Streaming with Remote Rendering
Chapter 7

Conclusions

7.1 Summary and conclusions

In this thesis, we perform analysis, simulation, emulation and build 3D video streaming prototypes for multiview 3D video. As the result of this work, we come to the following concluding statements for each individual chapter. Afterwards, we will discuss the research questions.

Chapter 2 provides an overview of efficient 3D video data-representations and proposes a novel rendering algorithm. Our region-based all-in-focus rendering algorithm allows to render a bandwidth-reduced version of a light-field representation at a quality that is visually indistinguishable from the original views. The algorithm achieves this by employing image segmentation to maximize the rendering quality at depth discontinuities in the scene. This is demonstrated with an experimental analysis of the impact of scene-depth discontinuities on the all-in-focus rendering quality of our algorithm. In our experiments, we measure the rendering quality using publicly available data sets and compare it with the best known all-in-focus rendering algorithm [64]. Our proposed algorithm improves the rendering quality both quantitatively (RMSE reduction of 10% on average) and, more significantly, visually by a noticeable reduction of the rendering blur, producing sharp renderings in the same regions.

In Chapter 3, we propose a layered streaming framework as an efficient solution to resource heterogeneity and argue for the benefits of this framework. To support this claim, we have built a streaming prototype according to that layered framework. The prototype uses an efficient 3D video representation in the form
of texture and disparity map and achieves a good video quality and real-time performance without a need for specialized networks or hardware. The difficulty at the time of development was a lack of available software for virtual-view rendering, as well as a unifying infrastructure for multi-stream transmission. Due to its support for stereoscopic video, our prototype is acknowledged as one of the first two stereoscopic-streaming prototypes in the research community [49]. It should however be noted that, to the best of our knowledge, this is also the first streaming prototype to support multiple-perspective viewing with a virtual-view rendering functionality. An architectural finding of this work is that our use of layers allows to scale the rendering quality with resource availability and to support selective view streaming with interactive feedback.

In Chapter 4, we propose an adaptive streaming algorithm that achieves a continuous streaming and high-quality multiview 3D rendering over best-effort networks, despite the time-varying available bandwidth. The algorithm performs a virtual-view streaming adaptation, using an optimized joint texture-depth rate allocation. We demonstrate that gains of up to 2 dB in average video quality are possible over conservative, non-adaptive streaming strategies. Our results also demonstrate gains of up to 0.7 dB are achievable with the proposed joint texture-depth rate allocation, compared to an allocation that considers the texture and depth streams as equally important for the rendering quality. This algorithm is acknowledged in the community as the first algorithm for adaptive 3D video streaming that performs a joint optimization of the 3D data representation, its rendering algorithm and the compression algorithm [53]. The proposed allocation maximizes the adaptive 3D video streaming quality in the sense of the spatial and temporal quality of rendered virtual views. The spatial quality is optimized with a joint rate allocation of texture and depth streams related to the same virtual view. The temporal optimization adapts the computed allocations to minimize the distortion variation and thus quality changes when streaming over time-varying transmission channels.

In Chapter 5, we present a video-prefetching algorithm that adapts to both the available bandwidth and the user-navigation patterns. We show that our algorithm achieves a low interaction latency and high rendering quality in 3D video streaming systems with large delay and time-varying available bandwidth. This novel algorithm achieves these properties by utilizing the available bandwidth efficiently and optimizing its streaming rate allocation to provide quality-smoothed view transitions. We have experimentally validated the correctness of the adapta-
tion to user-navigation patterns and demonstrated that the algorithm is capable of smoothing the view transitions in the PSNR sense. In addition, we have provided visual evaluation to show that visually disturbing artifacts are avoided when switching views.

In Chapter 6, we argue that by regarding the 3D video streaming system as a distributed application and offloading the view-rendering functionality to a remote location, we can potentially reduce the bandwidth requirements from linear in the number of streams to a constant. We support this claim by implementing a prototype that emulates this delivery scenario. Further, we argue by analysis that providing the view-rendering computation as a service of the existing content-delivery networks is both technologically possible and useful for cost optimizations in multiview 3D video streaming systems. To the best of our knowledge, this is the first multiview 3D video streaming proposal to consider such a distributed system architecture.

7.2 Discussion on research questions

Research question 1

With respect to the research question on bandwidth-efficient rendering algorithms and associated 3D video data-representations, we provide the following general conclusions. First, the proposed algorithm for all-in-focus light-field rendering uses an undersampled light-field representation of a 3D scene to render virtual views at a good visual quality. The novelty of the algorithm is in its use of selective region-based image segmentation as an intermediate step, which allows to constrain the subsequent step of template matching to the segmented regions, thereby minimizing rendering artifacts. We experimentally show that the proposed algorithm is bandwidth-efficient, in the sense that it renders virtual views of the given scene, and outperforms the state-of-the-art algorithm both quantitatively and visually (Table 2.1 and Figure 2.13 in Section 2.1.2), while using the same original views and geometry information. Second, we present two alternative algorithms for a comparison, that rely on scene-geometry models to reduce the scene sampling rate. We provide literature references to original papers, where it is shown that the disparity-based algorithm from Section 2.2.4 and the depth based image rendering (DIBR) algorithm from Section 2.2.3, are also bandwidth-efficient in the sense of the previously mentioned definition.
The results achieved with the three above rendering algorithms show that, by employing an efficient 3D video data-representation and a bandwidth-efficient rendering algorithm, it is possible to significantly reduce the bandwidth requirement of a 3D video streaming system at a negligible penalty in rendering quality (all three algorithms are bandwidth-efficient). It is important to note that our work does not perform a direct comparison between different 3D video data-representations with respect to their efficiency when sampling a given scene (e.g., light field-based and geometric model-based representations). Therefore, although interesting, the question of finding the most efficient representation for the given scene is beyond the scope of our research question. However, just for discussion, we note that the image based rendering literature regards the efficiency of a light field representation as inferior to that of depth-based representations [12, 13]. Unfortunately, the literature often makes unrealistic assumptions about depth image based rendering systems. This includes assuming the availability of algorithms or hardware for accurate depth estimation, irrespective of the scene complexity, or the availability of view-rendering algorithms that can handle occlusions and reflective surfaces, irrespective of the scene complexity. Both assumptions are entirely open issues in depth image based rendering, as illustrated by recent research that is published after our research work was already completed [209, 210].

Research question 2
We have addressed the question on providing a unifying solution to heterogeneity in 3D video streaming systems as follows. First, our choices of the layered streaming model and of the efficient disparity and depth-based representations to use with this layered model, allow the streaming system to accommodate receivers that have highly heterogeneous resources. The layered streaming model enables the system to scale the rendering quality with resource availability, which we demonstrate with experiments using displays with and without stereoscopic 3D rendering resources. In addition, the heterogeneity of viewing preferences is also accommodated in our system through our use of selective view streaming with interactive feedback. This solution enables each user to explicitly specify the desired viewing position and receive only the texture and disparity streams required to render a virtual view from that position. Second, we show that by implementing the layered streaming model inside a novel, remote-rendering architecture, a particularly challenged class of heterogeneous receivers, namely those with severe bandwidth- or cost-limitations, can also be supported in the system.
More specifically, the concept of layered streaming and the unifying aspect of it has been spread over two chapters (Chapter 3 and Chapter 6). In Chapter 3, we present and validate the concept, whereas in Chapter 6 we present an extreme case where multiview 3D video viewing is enabled at receivers using only one layer, by using our remote-rendering solution. Therefore, our architectural contribution demonstrates the adequacy of the proposed layered streaming framework as a unifying solution to resource heterogeneity.

Our experimental results are limited in that, due to time constraints, we could not validate our layered streaming model and the remote-rendering architecture using all 3D data-representations considered in this thesis, most notably, the light field representation. Due to similar constraints, we could not validate the entire remote-rendering and streaming architecture, including a functional implementation of a 3D streaming-CDN. Nevertheless, the performed experiment has captured the essentials of the proposed remote-rendering model and has illustrated its feasibility. Furthermore, the underlying streaming-CDN implementation challenges and solutions are well understood. As a result, additional experiments are likely to confirm our main conclusions, namely that efficient 3D video streaming systems should employ a layered streaming model and that they should consider remote view rendering as a useful concept.

**Research question 3**

With respect to the research question on *algorithms for bandwidth-efficient and low-latency 3D video streaming*, we provide the following discussion and conclusions. We have shown that it is possible to design adaptive streaming algorithms that simultaneously satisfy the requirements for a continuous and low-latency multiview 3D streaming. Further, we have demonstrated the efficiency of our multiview 3D streaming solutions in realistic scenarios, characterized by time-varying available bandwidth and large delays. In particular, our algorithms for minimizing the distortion variations in virtual views and user-adaptive prefetching optimize their utilization of the available bandwidth, by maximizing the perceived rendering quality in the sense of spatial and temporal quality of rendered virtual views. We have demonstrated these properties by providing, where possible, a comparison with state-of-the-art solutions and a quantification of the achieved gains.

We note that the accuracy of our conclusions is limited in quantitative terms, by our choice of metrics and models in the following domains. First, we use MSE
7. Conclusions

(PSNR) as a quality metric for rendered virtual views, instead of a metric based on human perception. Similar to our work, recent work on the quality trade-offs between texture and depth in video coding does not employ perceptual metrics and is limited to experimental studies of specific coding distortion-induced artifacts in virtual views [211, 212]. Second, we have not fully characterized the robustness of our algorithms to model errors, most notably, the bandwidth constancy model used for available bandwidth estimation and the user-interaction model employed to optimize the prefetching. However, our algorithms can easily be modified to incorporate more robust models and better video quality metrics in the future.

More importantly, we expect that the main value of our work is that our models and algorithms will be used as elements in any multiview 3D streaming system in the future. In fact, a number of our original contributions and design decisions is currently finding its way into state-of-the-art streaming solutions and research methodologies, as evidenced by recent research that is published after our work has been completed. Specifically, this applies to the following elements of our work: our use of a layered scene description for free-viewpoint video applications (MPEG 3DV [213]), our use of the MSE (PSNR) as a quality metric for rendered virtual views [214, 215], our choice of a 3D data representation to support heterogeneous displays in a unifying way (MPEG 3DV [213]), our optimization of the rates of texture and depth streams so as to maximize the quality of rendered virtual views in multiview video coding (MPEG 3DV [213]) and streaming [214, 215], our decision to measure the streaming quality of virtual views in the sense of MSE differences between the views rendered from raw streams and the views rendered from decompressed streams [214], our decision to explicitly control the variation of streaming quality over time [26], our bandwidth adaptation that considers the details of the underlying congestion-controlled connection [216, 217, 36], our decision to design the adaptation algorithm such that buffer starvations are minimized first and quality is optimized second [26], and our decision to design the prefetching algorithm such that a low interaction latency is ensured first and quality is optimized second [218]. These recent references support our main conclusion, namely that efficient 3D video streaming systems should employ adaptive streaming algorithms and a rate allocation that jointly considers the constituent components of its 3D data-representation, the coding algorithm and the rendering algorithm.
7.3 Future work

This thesis is addressing original work on constructing resource-efficient 3D video streaming systems. We believe more research work is needed for the applications based on this technology to become commonplace in the future Internet. We propose several directions for future work based on the findings of this thesis.

1. **Advances on efficient 3D video data-representations and rendering algorithms.** This thesis presents algorithms that efficiently render 3D video data-representations with acceptable quality. However, more work is needed to ensure that the rendering quality matches today’s high-quality 2D video and that it does so at an affordable cost. Additionally, it is important to compare the efficiency of different 3D video data-representations (e.g., light field and geometry-based) when sampling a given scene. For a large class of scenes, a hybrid 3D video data-representation may offer significant advantages. If we pursue such hybrid representations, hybrid rendering algorithms that cut across boundaries will be needed, in our case between light-field rendering, disparity-based rendering and depth image based rendering.

2. **Joint selection and optimization of 3D video data-representations and compression techniques.** The joint texture-depth rate allocation algorithms presented in this thesis achieve good video quality when the 3D data representation is encoded at a high quality. With low-quality compression, artifacts due to texture and depth quantization errors are visible. Therefore, the development of new allocation algorithms will need to be accompanied with additional work on human perception-based stereoscopic-[219, 220] and streaming-video quality metrics [26]. Recently, new work has been published on depth perception in stereoscopic 3D video and coding artifacts. This discussion on perception and quality metric has not yet converged to a broadly accepted approach and specification. Additionally, the multiview 3D video case is even more complicated in terms of perception.

3. **Adaptive streaming algorithms.** This thesis makes the first steps towards efficient 3D video streaming by defining algorithms adaptive to both the available bandwidth and the user-navigation patterns. Based on the experience gained, we observe a need for additional work in the following domains. First, a better characterization of user-navigation patterns will be important to further improve the adaptive prefetching performance. Second, more
7. Conclusions

experiments with sequences of different complexities (including motion and depth variability), will be needed to fully characterize the achievable gains of the proposed algorithms.

4. Remote-rendering architectures and algorithms. Chapter 6 proposes an architecture with view rendering as an extended functionality of the existing streaming-CDNs. We note that this proposal is based on our paper published in 2007 [51] and reflects the then state-of-the-art in distributed systems. As some Cloud-computing researchers have recently pointed out [92, 221], most future applications are likely to be distributed, consisting of components executed in the Cloud and components executed locally. In 2007, we argued that such a distributed model will be beneficial for future 3D video streaming systems. The system infrastructures to make this vision a reality are being widely deployed today. We therefore believe that an important area of future research is to actually build a complete 3D video streaming service according to this model.

More generally, our work is the first to provide algorithmic and architectural solutions for efficient multiview 3D video streaming, by optimizing the choice of a 3D video data-representation, its rendering, streaming and compression algorithms, in an integrated fashion. Therefore, our contribution is the first step towards the goal of achieving an efficient and high-quality multiview 3D video streaming in the future Internet. To achieve this efficiency goal, further progress on this topic will require to depart from simplistic assumptions often found in the literature. We believe that the underlying reason is a lack of common understanding between the research communities in the areas of compression and rendering on one end and streaming and networking on the other end. It is evident that good streaming proposals will jointly exploit both research areas. Because of the relevance of bandwidth costs of 3D video streaming, joint work on transport, encoding and rendering will be needed to make efficient multiview 3D video streaming a reality.
Bibliography


[196] Microsoft Corporation, “Advanced systems format (ASF) specification,”  
Revision 01.20.03, Dec. 2004.

[197] “Four reasons we choose Amazon’s Cloud as our computing platform,”  


[199] “Encoding.com,”  
http://www.encoding.com/.

[200] “Zencoder,”  
http://zencoder.com/.

[201] “Onlive,”  
www.onlive.com/.

[202] “Expanding the Cloud - adding the incredible power of the Amazon EC2 cluster GPU instances,”  

[203] N. Satish, M. Harris, and M. Garland, “Designing efficient sorting algorithms for manycore GPUs,”  
in Proc. 23rd IEEE Parallel & Distributed Processing Symposium, 2009.

[204] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands,  
K. Keutzer, D. A. Patterson, W. L. Plisker, J. Shalf, S. W. Williams,  
and K. A. Yelick, “The landscape of parallel computing research: A view from Berkeley,”  

“Global 1-Mbps peer-assisted streaming: Fine-grain measurement of a configurable platform,”  

Epema, M. Reinders, M. van Steen, and H. Sips, “Tribler: A social-based peer-to-peer system,”  
in IPTPS, 2006.


Curriculum Vitae

Goran Petrovic obtained his Dipl.Ing. degree in Electrical Engineering and Telecommunications from University of Nis, Serbia. His graduation thesis focused on the design and implementation of VDSL systems. He received his M.Sc. in Communications Technology from University of Ulm, Germany, where his work centered around error control in standards-based wireless video (MPEG-4) and multimedia signaling protocols (IETF SDPng). He started his research on algorithms and architectures for efficient 3D video streaming in 2006, as a Ph.D. student at Eindhoven University of Technology and a member of Freeband I-Share project. He continues to work on 3D and multiview video as a research assistant with University of Saarland, Germany, and a principal investigator at Intel Visual Computing Institute (IVCI) in Saarbruecken, Germany. He won Best Graduation Thesis award from the University of Nis and Siemens scholarship for his M.Sc. studies. He published 15+ papers and gave several research talks on multiview streaming in academia and industry research labs. He is a member of technical program committees for NEM Summit and IEEE ICCE-Berlin. He also serves as a reviewer on topics related to video streaming and coding for leading international journals and conferences, including IEEE Transactions on Circuits and Systems for Video Technology and ACM Transactions on Multimedia Computing, Communications and Applications.
Acknowledgments

This dissertation would not have been possible without my informal advisors: my mother Dragana, my father Miomir and my uncle Gojko. It is your unconditional love, selfless support and mentorship that have brought me to the end of the Ph.D. process. I owe this dissertation to you.

I thank my promotor Prof. Dr. Peter H. N. de With, for providing a continuous funding for my work and for his critical comments that improved the presentation quality of my dissertation and earlier papers. I also thank all my committee members for their time and willingness be a part of my dissertation defense. The core members of the committee, Prof. Dr. Inald Lagendijk, Prof. Dr. Thorsten Herfet, Prof. Dr. Johan Lukkien and Prof. Dr. Antonio Liotta, deserve my additional gratitude for their extensive feedback and detailed comments. Special thanks goes to Prof. Dr. Thorsten Herfet of Saarland University, whose funding has enabled me to continue my streaming research after leaving Prof. de With’s VCA group in Eindhoven. During the last stages of the dissertation process, I have also received a lot of administrative help from Anja van der Valk, including scheduling of the review appointments and accompanying me all the way to the thesis defense.

I would also like to express my gratitude to a number of colleagues in Eindhoven who contributed their time, skill and effort in order to help me perform the work described in this dissertation. My experimental results on 3D video streaming would not have been possible without Dirk Farin and Luat Do, who have designed and implemented the disparity-based and the depth image rendering algorithm, respectively. The light field rendering algorithm was co-designed with Aneez Kadermohideen Shahulhameed. The value of my work on building the streaming prototype was greatly enhanced by Shudong Chen and Richard Verhoeven, who integrated the prototype in their large streaming demonstrator. Several
Acknowledgments

experiments in this dissertation were based on YUV processing tools written by Ivo Creusen. I also thank all my paper co-authors for their ideas and commitment, as this dissertation is based on a number of previously published conference papers.

I additionally thank a number of colleagues and friends, whose research excellence has served as a model for me and whose wonderful personalities have made a long-lasting impact on me. Thank you all members of VCA lab in Eindhoven, in particular: Egor Bondarev, Bahman Zafarifar, Prarthana Shrestha, Jungong Han, Ping Li, Yannick Morvan, Rob Albers, Rob Wijnhoven, Chrysiz Papalazarou, Marijn Loomans, Weilun Lao, Milan Pastrnak, Sacha Cvetkovic, Lykele Hazelhoff, Julien Vijverberg, Sveta Zinger. Thank you all members of NT lab in Saarbrücken, especially: Jochen Miroll, Manuel Gorius, Zakaria Keshta, Yongtao Shuai, Christopher Haccius and Zhu Li. Thank you all members of Freeband I-Share project. Thank you all members of Signal Processing Systems department in Eindhoven. A special thanks to my earlier formal and informal advisors in Nis and in Ulm, including: Prof. Dr. Dragan Draca, Prof. Dr. Milena Stankovic, Prof. Dr. Andreas Kassler, Prof. Dr. Martin Bossert, Prof. Dr. Peter Schulthess, Prof. Dr. Mihajlo Stefanovic and Prof. Dr. Veljko Milutinovic.

Last but not least, I thank my friends, without whom it would have been impossible to maintain a good balance between work and play. I thank those in Germany (Marija and Frederic, Kazimir and Miriam), in The Netherlands (teta Mira, Zoran and Ankica, Milenko and Brankica, Marija and Rowin, Dragan and Milena, Dusan and Aleksandra, Igor and Jelena, Sneska and Henk), in Americas (Lena, Dragan, Mira, Rade, Dusan, Mirce, Enrique, Juan, James, Borko) and all my Serbian relatives and friends, in which case an exhaustive list is almost certainly better left for future work (Momir, Rupi, Pesa, Vlada, Luta, Jelena, Vesna, Stefi, Marija, Toske, Kosta, Martin, Milan, Bata, Dragan, Julia, Robert, Nebojsa...).

I dedicate this dissertation to my daughter Anja, my wife Ana and my late grandparents Margita and Momir Sinobad.

Goran Petrovic